FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

3D Uterine Cavity Reconstruction for Computer-Assisted Hysteroscopy

Ana Filipa Pereira Vieira da Rocha Fernandes

Mestrado em Bioengenharia

Especialização em Engenharia Biomédica

Supervisor: Prof. Dr. António Pedro Rodrigues Aguiar

Collaborator: Daniel Corona Oliveira Costa

June 24, 2025

3D Uterine Cavity Reconstruction for Computer-Assisted Hysteroscopy

Ana Filipa Pereira Vieira da Rocha Fernandes

Mestrado em Bioengenharia

June 24, 2025

Abstract

Hysteroscopy is the gold-standard technique for diagnosing and treating intrauterine abnormalities such as polyps, but the lack of depth perception in standard two-dimensional (2D) monocular imaging systems limits spatial understanding during the procedure. This dissertation explores the feasibility of generating three-dimensional (3D) reconstructions of the uterine cavity from monocular hysteroscopic video, with the aim of improving diagnostic accuracy and clinical navigation.

To this end, a modular depth-based reconstruction pipeline was developed, adapted from the *EndoSLAM* framework. The solution focuses on monocular depth prediction using convolutional neural networks, followed by real-time-compatible 3D reconstruction. Due to the lack of available gynecological datasets, two sources were used: the *EndoSLAM* stomach subset, chosen for its visual similarity to hysteroscopic scenes, and a custom dataset generated with a 3D-printed uterus phantom. While neither dataset allowed for full quantitative validation, the reconstruction pipeline produced spatially coherent models under controlled conditions.

A major contribution of this work was the optimization of the depth estimation module to improve inference speed. Techniques such as batch processing, input downscaling, and OpenCV-based preprocessing reduced inference time by nearly 50%. When integrated with the reconstruction pipeline, the system achieved approximately 38 frames per second, meeting real-time constraints for hysteroscopic video streams. However, full online execution remains limited by the absence of real-time pose estimation.

In conclusion, this work demonstrates the technical viability of real-time 3D uterine cavity reconstruction from monocular video and establishes a foundation for future extensions. Potential improvements include integrating visual odometry for online pose estimation, enhancing depth prediction through domain-specific training and temporal consistency, and improving dataset quality. These developments could, together, support the creation of tools for computer-assisted hysteroscopy that can be used in clinical practice.

Keywords: Hysteroscopy, Monocular Depth Estimation, 3D Reconstruction, Deep Learning

Resumo

A histeroscopia é a técnica padrão para o diagnóstico e tratamento de anomalias intra-uterinas, como pólipos, mas a falta de perceção de profundidade nos sistemas de imagem monocular 2D padrão limita a compreensão espacial durante o procedimento. Esta dissertação explora a viabilidade de gerar reconstruções tridimensionais (3D) da cavidade uterina a partir de vídeo histeroscópico monocular, com o objetivo de melhorar a precisão do diagnóstico e a otimização dos cuidados.

Para o efeito, foi desenvolvido um *pipeline* de reconstrução modular baseado em mapas de profundidade, adaptado da estrutura *EndoSLAM*. A solução centra-se na previsão de profundidade monocular, utilizando redes neurais convolucionais, seguida da reconstrução 3D. Devido à falta de dados ginecológicos disponíveis, foram utilizadas duas fontes: o subconjunto de estômago *EndoSLAM*, pela sua semelhança visual com imagens histeroscópicas, e um conjunto de dados personalizado gerado com um fantoma de útero impresso em 3D. Embora nenhum dos conjuntos de dados tenha permitido uma validação quantitativa completa, o *pipeline* de reconstrução produziu modelos espacialmente coerentes em condições controladas.

Uma das principais contribuições deste trabalho foi a otimização do módulo de estimativa de profundidade, com o objetivo de melhorar a velocidade de inferência. Técnicas como processamento em lote, redimensionamento da entrada (*downscaling*) e pré-processamento com OpenCV reduziram o tempo de inferência em quase 50%. Quando integrado ao *pipeline* de reconstrução, o sistema alcançou aproximadamente 38 quadros por segundo, cumprindo os requisitos de tempo real para fluxos de vídeo histeroscópicos. No entanto, a execução totalmente *online* continua limitada pela ausência de estimativa de pose em tempo real.

Em conclusão, este trabalho demonstra a viabilidade técnica da reconstrução 3D da cavidade uterina em tempo real a partir de vídeos monoculares, estabelecendo uma base sólida para desenvolvimentos futuros. As melhorias potenciais incluem a integração de odometria visual para estimativa de pose *online*, o aperfeiçoamento da previsão de profundidade através de treino específico ao domínio e da consistência temporal, bem como a melhoria da qualidade do conjunto de dados. Estes avanços poderão, em conjunto, apoiar a criação de ferramentas para histeroscopia assistida por computador com potencial de aplicação na prática clínica.

Palavras-Chave: Histeroscopia, Estimativa de Profundidade Monocular, Reconstrução 3D, Deep Learning

Acknowledgements

Firstly, I would like to thank my supervisor, Prof. Pedro Aguiar, and my co-supervisor, Daniel Corona, for their constant guidance, availability, and openness in allowing me to develop this project. Their knowledge and support were instrumental throughout this journey.

I would also like to dedicate all my efforts to the constant strength that my whole family has tirelessly given me. Mother, father and siblings, I am now and will always be very grateful for all the love and dedication/support you have shown me over the years. To Ismael, thank you for believing in me, encouraging me, and standing by my side. Furthermore, I am grateful for the endless and tireless love of my pets, who, unaware of my difficulties, were there for me every day. Finally, I would like to thank my friends, without whom I would not be here, and who always gave me words of motivation and confidence.

Special thanks to technician Pedro Alves, student Bruno Santos, and the researchers at the SYSTEC Laboratory (Centro de Investigação em Sistemas e Tecnologias) for their invaluable help with the silicone model extraction, use of the robotic arm, and 3D printing, respectively. Finally, I must thank all the teachers and the university, who gave me the tools I needed for my transformation over these five years.

Thank you all from the bottom of my heart. I have no words other than to express my deepest appreciation to everyone who has been present.

Filipa

"Believe it or not,
I can actually draw."

Jean-Michel Basquiat

Contents

List of Figures

List of Tables

Abbreviations

2D	Two-dimensional
3D	Three-dimensional
GT	Ground Truth
RGB	Red, Green, Blue
CT	Computed Tomography
SLAM	Simultaneous Localization and Mapping
CNN	Convolutional Neural Network
MAE	Mean Absolute Error
MSE	Mean Squared Error
RMSE	Root Mean Squared Error
PSNR	Peak Signal-to-Noise Ratio
SSIM	Structural Similarity Index
STL	Standard Tessellation Language
FPS	Frames Per Second

Chapter 1

Introduction

This chapter introduces the clinical context and motivation for the work presented in this thesis, outlines the main objectives of the project, and provides an overview of the structure of the remaining chapters.

A polyp is a growth of tissue that protrudes from a surface in the body, usually a mucous membrane, and can develop in a variety of areas. They are most commonly found in the uterus or colon, but can also grow in places such as the ear canal, nose, throat, cervix, stomach, rectum and bladder [2].

When polyps form inside the uterus, they are called uterine or endometrial polyps and are among the most frequent abnormalities found in this region. These tissue overgrowths, composed of glands, fibrous stroma, and blood vessels, can vary in size, shape, and number. They may be firmly attached to the uterine lining or connected to it by a thin stalk [3, 9]. Furthermore, these polyps may present with symptoms or remain entirely asymptomatic, and even though they are usually benign, there is a small risk of malignancy [22].

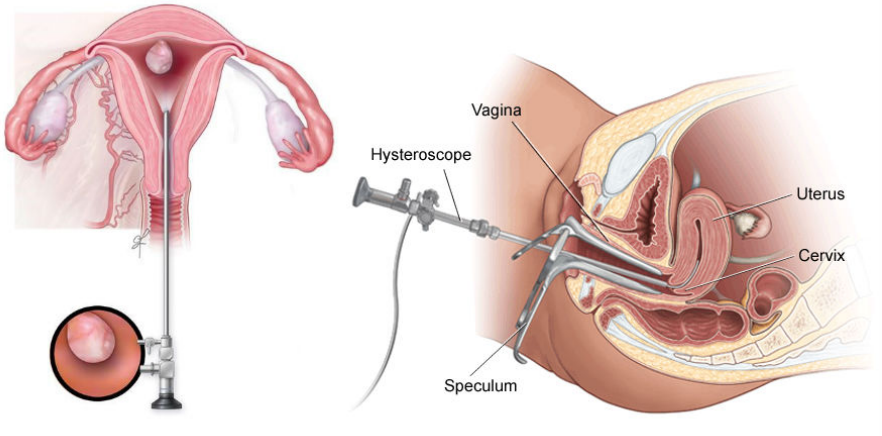


Figure 1.1: Illustration of a hysteroscopy procedure, from [17].

The method of choice to detect and treat these anomalies has become hysteroscopy. As illustrated in Figure 1.1, this minimally invasive technique involves inserting a thin, lighted camera — known as a hysteroscope — through the cervix to directly visualize the inside of the uterus.

In recent years, hysteroscopy has become an increasingly central tool in gynecological care, especially for identifying and managing intrauterine conditions such as polyps, fibroids, and adhesions [11]. Its ability to provide real-time visualization while avoiding more invasive surgical procedures makes it a preferred option among clinicians. Contributing to this, the evolution of office-based operative hysteroscopy has significantly improved accessibility and many procedures can now be performed comfortably in an outpatient setting in less than 30 minutes, often without the need for general anaesthesia. This shift towards a "see and treat" model has proven to be both time-efficient and cost-effective, while also contributing to greater patient satisfaction and faster recovery times [32].

1.1 Impact On Women's Lives

Although endometrial polyps are often benign, their impact on women's lives can be far from negligible. Many women with polyps experience symptoms such as irregular menstrual bleeding, pelvic discomfort, or infertility, conditions that not only affect physical health but also emotional well-being [3, 9]. For example, if a person actively trying to conceive has a uterine polyp, that can bring uncertainty, frustration, and in some cases, repeated pregnancy loss. These experiences can lead to significant psychological distress, particularly when the cause of infertility remains undiagnosed for a long period [41].

Even when asymptomatic, polyps still raise concerns due to their potential for malignant transformation [22]. This possibility, combined with their tendency to go unnoticed, highlights the importance of timely and accurate diagnosis. Therefore, early detection not only provides relief from symptoms but also helps to exclude more severe conditions through biopsy or surgical extraction.

It is also important to notice that the prevalence of endometrial polyps appears to increase with age. A large Danish population study found that while only 0.9% of women under 30 were diagnosed with polyps, this number climbed steadily with age, reaching a peak in the 40–49 age group [10]. This trend suggests a strong hormonal or age-related component in polyp development. At the same time, studies show that many polyps go unnoticed, especially in younger women, since they may not produce symptoms [10].

Despite the fact that the majority of polyps are benign, the risk of endometrial cancer, though relatively low, cannot be ignored. A research has shown that approximately 1.3% of women with endometrial polyps may develop cancer, while malignancy confined to a polyp is observed in about 0.3% of cases [42]. These data further emphasize the clinical importance of distinguishing between harmless and potentially harmful lesions, particularly in high-risk populations.

From a healthcare systems perspective, a 2021 study reported that over 60,000 hysteroscopies are performed annually in the United Kingdom, mostly to investigate abnormal bleeding and related gynecological conditions [37]. This growing demand reflects not only the prevalence of

such issues but also the increased reliance on minimally invasive approaches like hysteroscopy for accurate diagnosis and effective treatment. A recent U.S. market report (Figure 1.2) further illustrates the upward trend in hysteroscopic procedures, driven by the need to manage polyps, fibroids, infertility, and postmenopausal symptoms [13].

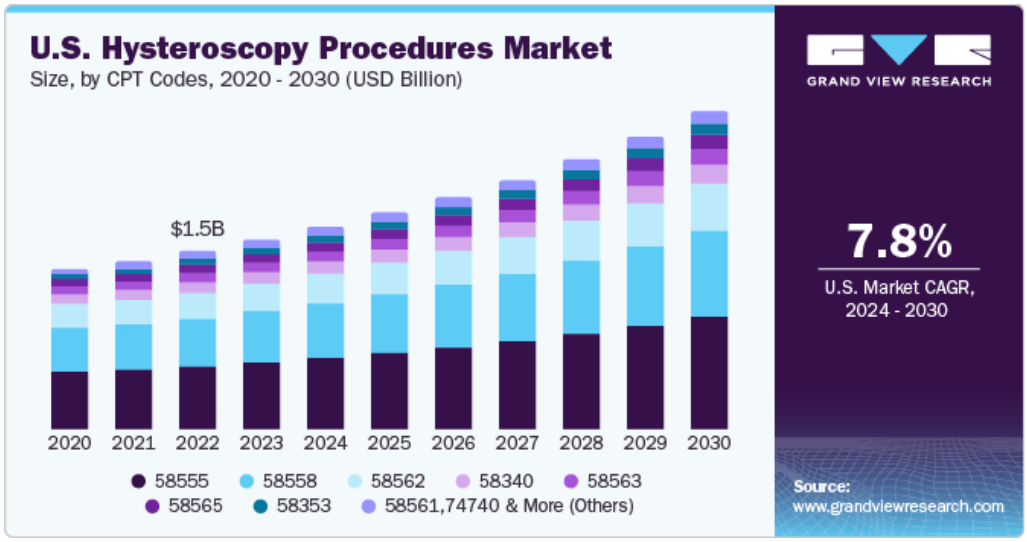


Figure 1.2: Hysteroscopy Procedures Market Trends, from [13].

Taken together, these findings show that while endometrial polyps may initially seem minor, their clinical and emotional impact is often substantial [3, 9]. Ensuring timely diagnosis and appropriate management is essential—not just to treat symptoms, but to improve quality of life and reduce long-term risks for women across different life stages [22].

1.2 Challenges in Hysteroscopy

Hysteroscopy is the most common intervention for the diagnosis and treatment of uterine abnormalities, including polyps. It enables healthcare professionals to visualize the uterine lining in real time, identify any anomalies, and undertake interventions, such as the removal of polyps. This technique has evolved significantly over the past two centuries and is now considered the gold standard for intrauterine evaluation [38].

Despite its utility, the procedure depends on the availability of specialized equipment, the presence of optimal conditions to ensure a clear and accurate visualization of the uterine cavity, and the expertise of the operator [32]. This is not only to execute the procedure, but also to interpret the resulting images, which can be subjective and affected by the person's experience, thus increasing the risk of misdiagnosis or missed abnormalities [40]. Another significant challenge is the reliance on high-quality imaging that can be compromised by anatomical variations, blood, or other obstructions within the uterine cavity [34].

Standard hysteroscopes are generally equipped with a single camera system, which captures a two-dimensional (2D) image [24]. Although these cameras provide adequate visualization for

most diagnostic and surgical procedures, the absence of depth perception makes it more challenging to assess spatial relationships within the uterine cavity [16]. Consequently, subtle abnormalities, such as small or hidden polyps, may be more difficult to detect.

Nonetheless, standard hysteroscopes remain the most affordable and accessible option. In contrast, stereoscopic hysteroscopes offer enhanced imaging with depth perception and increased detail, but are often more expensive and complex to use [34]. This creates a technology gap, i.e. the need for a cost-effective solution that bridges the benefits of advanced imaging while remaining practical for a wider range of healthcare providers [40]. By addressing these challenges, advances in imaging and computational techniques have the potential to improve diagnostic accuracy and patient outcomes [31].

1.3 Objectives

The aim of this dissertation is to study and contribute to the development of an image-based solution to assist surgeons during hysteroscopic procedures, reducing the risk that endometrial polyps are missed. This involves the depth prediction from two-dimensional (2D) images acquired via hysteroscopy, as well as the three-dimensional (3D) reconstruction of the uterine cavity to provide a comprehensive visualization of examined areas. The ultimate goal is to improve the diagnostic accuracy and patient outcomes through an innovative computer-assisted approach.

In essence, the objectives of the proposed project are as follows:

- Depth prediction: Design and implement approaches for predicting depth from monocular RGB hysteroscopic images, focusing on real-time performance and compatibility with clinical hardware;
- 3D reconstruction of the uterine cavity: Build a 3D reconstruction pipeline that uses predicted depth maps and known camera poses to generate a spatial model of the uterine cavity, allowing delineation of examined and unexamined areas;
- System integration and validation: Integrate the individual modules into a unified system and evaluate its performance using representative data to ensure robustness, accuracy, and usability in a clinical context.

1.4 Outline

In addition to the introduction, this document contains four other chapters. Chapter 2 presents a review of the state of the art, with a focus on monocular depth estimation and 3D reconstruction in endoscopic settings, particularly within the context of hysteroscopy. Chapter 3 describes the methodology, including dataset details, depth prediction using a CNN-based model, and the reconstruction pipeline for generating the 3D model. Chapter 4 presents the results obtained, accompanied by a comprehensive discussion of their implications and limitations. Finally, in Chapter 5, a conclusion is drawn, and future expectations are explored.

Chapter 2

State of the Art

Endometrial polyps represent a common form of uterine abnormality, affecting women in various age groups, especially those in their reproductive years and perimenopausal stages. These polyps develop within the endometrium, comprising glands, stroma, and blood vessels. Although they are generally considered benign, a small percentage may undergo malignant transformation, particularly in postmenopausal women, representing a potential risk factor for endometrial cancer [3]. These pathologies are often associated with symptoms such as abnormal uterine bleeding (AUB), pelvic pain, infertility, and recurrent miscarriage, which can have a significant impact on a woman's reproductive health and overall well-being [9]. For this reason, an accurate diagnosis of endometrial polyps is imperative for effective management and timely intervention.

Hysteroscopy is currently regarded as the gold standard for diagnosing and managing endometrial polyps [32]. However, most conventional systems employ monocular cameras that provide only 2D images. This limitation in depth perception may hinder accurate evaluation of spatial relationships within the uterine cavity, increasing the risk of incomplete diagnosis or misinterpretation [11]. Image quality can also be adversely affected by anatomical variations, intrauterine bleeding, or tissue obstruction, complicating lesion identification and assessment [32].

Despite these challenges, hysteroscopy offers the advantage of direct visual assessment and, in many cases, enables immediate therapeutic intervention. The ability to perform polypectomy during the diagnostic procedure streamlines patient care by combining evaluation and treatment in a single session[9, 32]. Nevertheless, the demand remains for more precise, reproducible, and accessible imaging tools that can enhance diagnostic accuracy, reduce operator dependency, and improve clinical outcomes.

Figure 2.1 presents a series of representative hysteroscopic frames, illustrating the visual variability encountered during navigation through the uterine cavity. The data was acquired using a standard hysteroscopic system at Centro Hospitalar Universitário São João (CHUSJ) in Porto, Portugal. These frames highlight typical challenges in intrauterine visualization, such as variable image clarity, anatomical complexity, and occlusions, which may complicate both diagnosis and treatment.

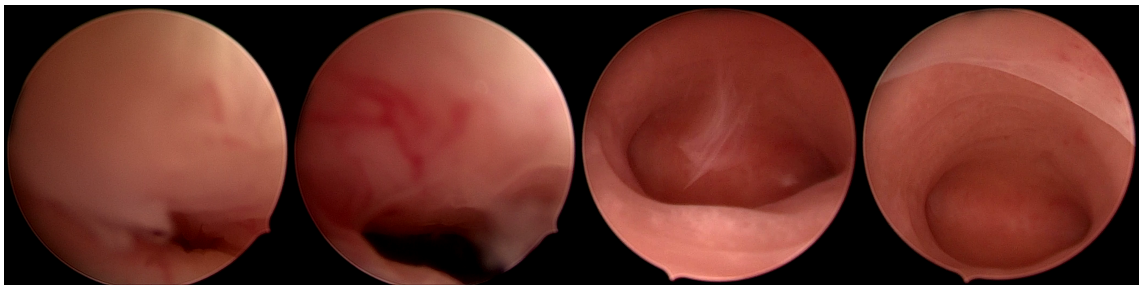


Figure 2.1: Example hysteroscopic frames captured at various positions along the intrauterine path.

2.1 Current Solutions for Uterine Cavity Examination

Over the years, various technologies have been developed to improve the visualization and assessment of the uterine cavity. These solutions can be broadly categorized into two areas: hysteroscopic systems used for real-time imaging, and post-processing techniques that enhance the diagnostic capabilities of hysteroscopic images, such as polyp segmentation and 3D reconstruction methods.

2.1.1 Overview of Hysteroscopic Systems

Hysteroscopy remains the primary method used in clinical practice for direct visualization and evaluation of the uterine cavity. It is considered a specialized form of endoscopy and is performed using a hysteroscope, a thin, lighted tube equipped with a camera, that is inserted through the cervix into the uterine cavity [36]. Hysteroscopes are typically classified as either rigid or flexible, depending on the shaft's structure and intended clinical use [38]. An example of a rigid continuous-flow hysteroscope is shown in Figure 2.2.

Figure 2.2: Example of a rigid continuous-flow hysteroscope (Karl Storz, 5 mm), commonly used in operative hysteroscopy (adapted from Vitale et al. [40]).

Rigid hysteroscopes are the most commonly used, particularly in hospital settings, due to their high image quality, mechanical durability, and compatibility with operative instruments. Many of these systems feature continuous-flow technology, which enables the inflow and outflow of distension media, typically saline. Continuous-flow systems are particularly beneficial in operative hysteroscopy, as they help clear debris and blood, ensuring sustained visibility during longer or more complex procedures. Furthermore, saline is a clear fluid used to gently expand the uterine cavity and maintain visualization throughout the procedure [40]. As a result, rigid hysteroscopes

are widely preferred for operative hysteroscopy and are also frequently used in diagnostics when detailed visualization is required [32].

On the other hand, flexible hysteroscopes are often favored in outpatient or office-based settings. Their flexibility allows easier navigation through the cervical canal and improved patient comfort, particularly in diagnostic procedures where anesthesia is not used. Although they differ mechanically, flexible hysteroscopes are similar in overall shape and design to rigid ones, typically featuring the same slender, tubular structure with a camera and light source at the distal end. Consequently, its their flexible shaft and smaller diameter that make these instruments especially suitable for office hysteroscopy and other minimally invasive approaches, where procedural efficiency and reduced discomfort are key priorities [32].

Both rigid and flexible hysteroscopes commonly use standard imaging systems based on a single camera, which capture 2D images of the uterine cavity. While adequate for detecting common abnormalities like polyps and fibroids, 2D systems lack depth perception — a limitation that can reduce spatial awareness and complicate the assessment of small or partially obscured lesions. Clinical studies confirm that rigid scopes generally offer superior optical quality compared to flexible ones, although both rely on 2D imaging systems by default [39, 40].

Hysteroscopes equipped with stereoscopic (3D) imaging systems — often referred to as stereoscopic hysteroscopes — provide improved depth perception and spatial accuracy by using binocular (dual-camera) setups. Although they do exist in experimental or high-end forms (e.g., dual-camera systems used in some rigid instruments), these systems remain rare in routine practice due to their high cost, specialized equipment, and the training required for effective use [40]. Even more so, integrating stereoscopic imaging into flexible hysteroscopes remains virtually nonexistent, as it involves complex engineering challenges and significantly increased equipment costs, along with limited spatial resolution and image quality [21].

Having that said, it is important to note that standard hysteroscopes remain the cornerstone of hysteroscopy, used in the majority of both diagnostic and operative procedures, regardless of whether the instrument is flexible or rigid. Therefore, efforts to enhance image quality and usability within these systems remain essential for improving diagnostic precision and clinical outcomes.

2.1.2 3D Uterine Reconstruction Methods

To enhance spatial perception during standard hysteroscopy, which typically provides only 2D monocular views, 3D reconstruction techniques have emerged as promising tools to augment conventional imaging.

It is important to note that, although stereoscopic hysteroscopy provides enhanced 3D visualization through the use of dual cameras, this is a 3D imaging technique rather than a 3D reconstruction method. Unlike stereoscopic systems, which rely on capturing 3D images directly, 3D reconstruction methods aim to generate a 3D model of the uterine cavity from 2D or monocular images. These methods are particularly valuable when stereoscopic systems are unavailable or impractical, such as in most routine procedures using monocular hysteroscopes [30].

Recent studies have increasingly explored the application of monocular 3D reconstruction techniques in endoscopic procedures. A particularly comprehensive and up-to-date overview is provided by Richter et al., who review state-of-the-art methods for generating 3D models from 2D endoscopic video [30]. Their work highlights the growing feasibility of real-time reconstruction in surgical environments, despite ongoing challenges related to computation and data availability. Although their focus is on gastrointestinal endoscopy, many of the reviewed techniques are directly applicable to hysteroscopic imaging, where similar constraints arise from monocular input, anatomical deformation, and low-texture surfaces.

The development of 3D reconstruction techniques for endoscopic procedures has gained increasing attention as a means to enhance intraoperative spatial understanding. Although most existing work focuses on gastrointestinal endoscopy, many of these methods can be adapted to other anatomical contexts, including hysteroscopy. The ability to generate a 3D model from monocular 2D video can facilitate improved navigation and more precise identification of abnormalities such as polyps or fibroids within the uterine cavity. However, due to anatomical deformation, limited texture, and constrained lighting typical of hysteroscopy, traditional triangulation-based methods face significant limitations [30]. As a result, alternative approaches such as Simultaneous Localization and Mapping (SLAM) and monocular deep learning have gained attention for their potential to enable accurate 3D reconstruction from standard hysteroscopic video.

SLAM Techniques

SLAM is a set of geometric techniques originally developed in robotics, where a system incrementally builds a map of an unknown environment while simultaneously estimating its own position within it [6]. The core principle behind SLAM is the joint estimation of structure (mapping) and motion (localization) from sequential image data. For further understanding, a general SLAM pipeline is illustrated in Figure 2.3.

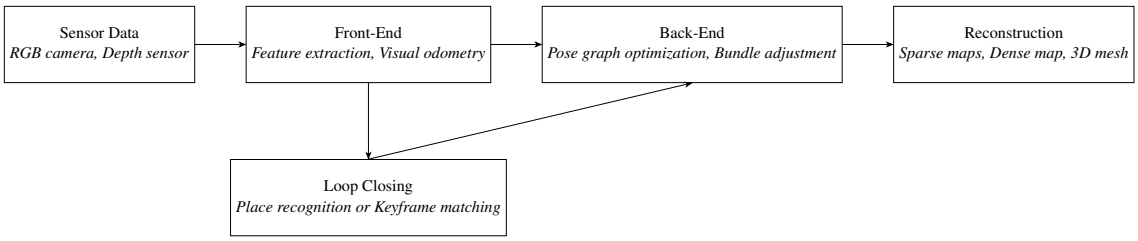


Figure 2.3: Custom SLAM pipeline diagram illustrating sequential processing and loop closure, with example components relevant to endoscopic 3D reconstruction.

One widely used category is visual SLAM, which operates on video streams captured by a camera. Within this, monocular SLAM systems — those relying on a single camera — have gained particular relevance due to their efficiency and compatibility with standard imaging setups [6]. These methods typically rely on geometric feature extraction and visual tracking to estimate the camera pose and incrementally reconstruct a sparse 3D map of the environment [30].

In the context of hysteroscopy, SLAM algorithms — particularly those based on monocular visual input — can be considered for reconstructing the 3D structure of the uterine cavity from standard 2D hysteroscopic video. While existing SLAM implementations, such as those in *EndoSLAM* [26], have been applied to gastrointestinal endoscopy, the underlying principles are adaptable to gynecological procedures as well, where similar imaging constraints apply. These approaches aim to support real-time intraoperative 3D modeling, providing surgeons with improved spatial context and navigational assistance.

However, SLAM techniques are not without limitations. For instance, they require substantial computational resources, which makes real-time application challenging [30]. Moreover, most SLAM algorithms assume a rigid environment and may struggle in deformable anatomical settings such as the uterus, further aggravated by low-texture regions and varying illumination conditions. Lastly, accurate tracking systems are also often needed, which may not always be feasible in routine clinical practice [26].

Deep Learning-Based Reconstruction

Deep learning-based 3D reconstruction is a rapidly emerging field. Recent advances have led to neural networks that generate 3D models from 2D endoscopic images, typically by inferring depth from sequences of monocular frames. These models typically use sequences of monocular frames to infer depth and build a spatial representation of the observed internal cavity.

A notable method is the use of self-supervised learning frameworks such as *Endo-SfMLearner*, which estimate depth and camera pose from monocular videos without requiring ground-truth labels. This approach has demonstrated real-time feasibility with acceptable accuracy in endoscopic applications [30, 26].

Another promising strategy involves Generative Adversarial Networks (GANs). For example, *pix2pix* — a conditional GAN originally proposed by Isola et al. for image-to-image translation — has been adapted to generate depth maps from monocular endoscopic images [19, 30].

In addition, Convolutional Neural Networks (CNNs), including architectures like *StereoNet* and *DispNetC*, have also been adapted for endoscopic depth estimation. Although originally designed for stereo inputs, these networks have been reconfigured for monocular applications and trained on large endoscopic datasets to generate dense depth maps in real time, an essential capability for accurate 3D reconstruction during surgery [30, 1].

Despite the promise of these deep learning techniques, the process requires substantial computational power and extensive training datasets, which may not always exist or be available. Besides that, ensuring real-time responsiveness continues to be a technical bottleneck.

EndoSLAM: Deep Visual Odometry and Depth Estimation

To overcome the limitations of classical SLAM in endoscopic environments, recent research has explored learning-based alternatives that estimate depth and camera motion directly from image data. A notable contribution in this area is *EndoSLAM*, which introduces a large-scale dataset

of endoscopic sequences, along with an unsupervised deep learning framework called *Endo-SfMLearner*. This model performs monocular visual odometry and depth estimation without requiring ground-truth labels, leveraging self-supervised learning losses to align predicted depth with image consistency and estimated motion [26].

Unlike traditional SLAM, which relies on handcrafted features and assumes scene rigidity, *EndoSLAM* is designed to handle the non-rigid, texture-scarce, and dynamic characteristics typical of medical settings such as hysteroscopy. As a result, it presents a promising direction for real-time 3D reconstruction under realistic clinical conditions.

Summary of Solutions for Uterine Cavity Examination

To consolidate the techniques discussed in this chapter, from traditional 2D hysteroscopic systems to advanced 3D reconstruction methods, Tables 2.1 and 2.2 present comparative overview of their core principles, advantages, and limitations. Although many of these methods were originally developed for gastrointestinal endoscopy, they provide adaptable technical foundations for hysteroscopic applications.

As the table illustrates, this transition from conventional monocular imaging to real-time 3D reconstruction reflects a broader evolution toward enhanced intraoperative spatial understanding. Yet, the clinical adoption of these advanced systems remains constrained by integration complexity, computational overhead, and domain-specific data limitations.

Table 2.1: Comparison of imaging methods relevant to hysteroscopy.

Method	**Description**	**Advantages**	**Limitations**
Standard Hysteroscopy (Monocular)	Uses a single camera to capture 2D images of the uterine cavity.	Widely available; low cost; simple to use.	Limited depth perception; less accurate for complex structures.
Stereoscopic Hysteroscopy (Binocular)	Dual-camera system enabling 3D perception. Rare in hysteroscopy.	Enhances depth perception; useful for spatial understanding.	High computational cost; limited availability in gynecology.

2.2 Datasets

As the primary aim of this work is to investigate and test techniques relevant to uterine anatomy, a thorough search was conducted for publicly available endoscopic datasets. However, to date, there are no publicly available, well-documented datasets focused on gynecological anatomy — particularly for hysteroscopy — that include the necessary ground truth data for 3D reconstruction tasks. Most available resources are instead centered on gastrointestinal organs, and although anatomically distinct, some of these regions share notable visual and structural similarities with

Table 2.2: Comparison of 3D reconstruction methods adapted from endoscopic applications.

Method	**Description**	**Advantages**	**Limitations**
SLAM (Simultaneous Localization and Mapping)	Applied in endoscopy to generate a 3D map while estimating the endoscope's motion.	Real-time 3D reconstruction; improves surgical navigation.	Requires substantial computational power; assumes rigid environments.
Deep Learning-Based 3D Reconstruction	Neural networks reconstruct 3D models from monocular endoscopic images.	Can handle monocular input; promising for real-time use.	Requires large datasets and computational resources.
Convolutional Neural Networks (CNNs, e.g., StereoNet, DispNetC)	Estimate depth from 2D endoscopic images using convolutional neural networks.	Real-time depth estimation; high accuracy with training.	Resource-intensive; performance may degrade in low-texture areas.
Generative Adversarial Networks (GANs, e.g., pix2pix)	Generate synthetic depth maps from monocular endoscopic images using adversarial training.	Produces realistic depth; adaptable to variable imaging.	Training instability; high data demands.
Self-Supervised Learning (e.g., Endo-SfMLearner)	Estimate depth and camera pose from monocular videos without labeled ground truth.	Eliminates need for GT labels; adaptable to medical data.	May require fine-tuning; accuracy sensitive to video quality.

the uterus, making them a practical alternative. For this reason, several datasets from the gastrointestinal domain were considered and evaluated for suitability in this context.

A particularly up-to-date and comprehensive overview is provided by Richter et al. in their review article *Advances in Real-Time 3D Reconstruction for Medical Endoscopy* [30], which was also referenced in the previous subsection. This work was used as a primary reference due to the quality of its analysis, and all datasets mentioned therein were reviewed as part of the selection process.

Only four of the datasets listed in that work were considered suitable, as they all meet key criteria: they provide monoscopic video sequences, contain dynamic scenes relevant to real-time reconstruction, and are fully available for use. These are *2D–3D Registration*, *Depth from Colon*, *EndoSLAM*, and *SimCol3D*. A summary of their core characteristics is presented in Table 2.3.

The remaining datasets were excluded either because they are based on stereoscopic imaging (e.g., Middlebury, KITTI, Phantom Cardiac), consist of static views only (e.g., SERV-CT), or are only partially available (e.g., Hamlyn).

The *2D–3D Registration* dataset (2021) provides real colonoscopy sequences paired with depth maps obtained via registration to Computed Tomography (CT)-derived 3D models [5]. The

Table 2.3: Comparison of selected datasets suitable for monocular 3D reconstruction tasks.

Dataset	**Organ(s)**	**Modality**	**GT Type**	**Notes**
EndoSLAM	Stomach, colon, small intestine	Real + synthetic	Structured light / simulated	Combines synthetic UnityCam and real ZED/RealSense sequences. GT varies by subset. Stomach data used in this work.
SimCol3D	Colon	Synthetic + real	CT-based / pseudo	Synthetic subset provides RGB, depth, and pose. Real subset lacks full GT annotations.
Depth from Colon	Colon	Synthetic	Simulated (Unity)	Generated from CT-based anatomy in Unity. Offers consistent RGB–depth alignment.
2D–3D Registration	Colon	Real	CT-registered	Colonoscopy videos registered to 3D models reconstructed from CT scans of the same patients.

Depth from Colon dataset (2019) consists of synthetic colonoscopy images and corresponding depth maps generated in a simulated environment, based on CT data [29]. Finally, *SimCol3D* (2024) includes both synthetic and real colonoscopy videos, offering labeled frames with RGB, depth, and camera pose information for its synthetic subset [28]. After reviewing these datasets in detail, it was confirmed that all three are exclusively focused on the colon, without anatomical diversity across the gastrointestinal tract.

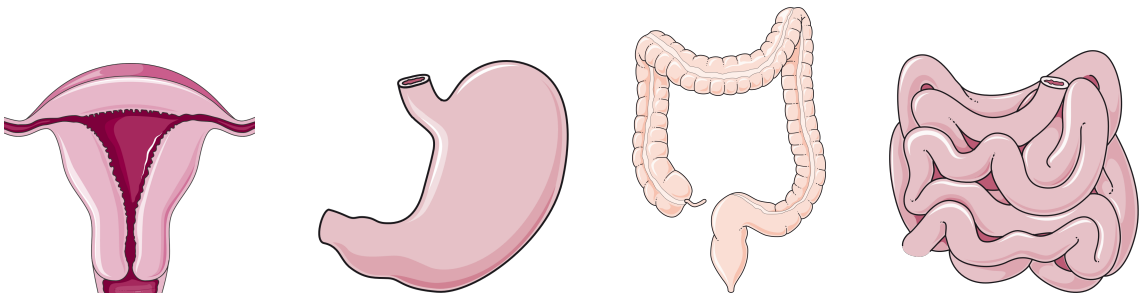
In contrast, the *EndoSLAM* dataset (2020) comprises annotated endoscopic sequences from multiple gastrointestinal organs, including the stomach, small intestine, and large intestine (colon) [26]. That said, the *EndoSLAM* dataset was ultimately selected, as it contains annotated sequences of the stomach, which is a region that, although distinct, shares several morphological and textural similarities with the uterine cavity. In the absence of gynecology-specific datasets, these similarities make it a reasonable proxy for initial testing.

Furthermore, *EndoSLAM* stands out for its comprehensiveness, offering RGB image sequences, corresponding ground truth camera poses, depth maps, and detailed metadata. It also includes an associated publication describing the acquisition setup, endoscope specifications, and sensor configurations [26]. This level of documentation makes it a strong and reliable resource for bench-

marking and validation.

To support the selection of the *EndoSLAM* dataset, in specific the stomach subset, for uterus-related experiments, it is important to clarify the basis of anatomical and visual similarity between these two organs. While not identical, the stomach and uterine cavity share several relevant characteristics: both exhibit rounded, enclosed geometries, feature mucosal tissue with relatively low-texture visual properties (when compared to the highly folded intestinal walls), and maintain similar reddish color palettes under endoscopic illumination. Furthermore, the endoscopic trajectories observed during navigation — involving gentle curvature and forward motion in a confined space — are also comparable.

Figure 2.4 provides a structural illustration of the uterus, stomach, colon, and small intestine, highlighting the closer geometrical similarity between the uterus and stomach. Additionally, Figure 2.5 presents a qualitative visual comparison between a real hysteroscopic frame of the uterine cavity (provided by Centro Hospitalar Universitário São João, CHUSJ) and a simulated gastroscopic frame from the *EndoSLAM* UnityCam data. Together, these figures help justify the use of stomach data as a visual and spatial proxy for early testing of monocular depth estimation and 3D reconstruction techniques.



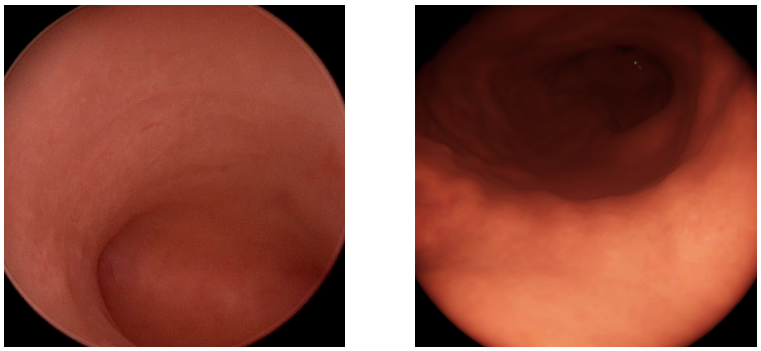
(a) Uterus (b) Stomach (c) Colon (d) Small Intestine

Figure 2.4: Comparison of relevant organ shapes adapted from Servier Medical Art [33], licensed under CC BY 3.0.

2.3 Proposed Solution

In response to the limitations identified in the current literature and clinical practice, this work proposes an image-based system for 3D reconstruction of the uterine cavity from monocular 2D hysteroscopic video. Inspired by the pipeline proposed in *EndoSLAM* [26], this approach focuses specifically on evaluating and applying monocular depth estimation techniques with an emphasis on achieving the real-time performance required for practical clinical use.

While the full *EndoSLAM* framework includes both depth and pose estimation via visual odometry, this study focuses exclusively on the depth estimation component. To substitute the pose estimation task, ground truth (GT) pose data is used throughout the implementation. Nonetheless, it is acknowledged that for a complete real-time intraoperative system, stable and efficient pose



(a) Hysteroscopic frame (uterine cavity)

(b) Simulated endoscopic frame (stomach)

Figure 2.5: Qualitative comparison between a real hysteroscopic frame (uterus) and a synthetic endoscopic frame (stomach) from the *EndoSLAM* UnityCam dataset. Despite anatomical differences, both scenes share similar visual features.

estimation would also be essential, and this will be fully considered in future work. By isolating and optimizing the depth estimation process, this study seeks to address one of the key computational bottlenecks and demonstrate the feasibility of real-time spatial mapping within the uterine cavity.

Having that said, *Endo-SfMLearner* architecture, originally trained on gastrointestinal endoscopic video, provides the conceptual foundation for the proposed method.

Unlike stereoscopic or traditional SLAM-based systems, which often require specialized hardware or impose computational burdens, this solution prioritizes real-time performance and broad clinical applicability for depth prediction. It is designed to operate on a standard clinical workstation, with computational tasks offloaded to an external computer connected via cable - a setup that ensures adaptability even in resource-limited environments and facilitates integration into routine practice without requiring significant hardware upgrades.

In conclusion, the core objective is to enhance spatial awareness during hysteroscopic procedures, allowing clinicians to systematically navigate and revisit areas within the uterine cavity. This 3D perspective can reduce the risk of missing critical lesions, particularly in low-visibility conditions. Ultimately, the goal is to assist in the earlier detection of endometrial polyps or other abnormalities during routine procedures, enabling timely treatment and thereby contributing to better patient outcomes.

Taken together, this work aims to demonstrate that real-time monocular depth estimation is feasible within the constraints of clinical hysteroscopy, serving as a stepping stone toward future AI-powered tools that could make diagnosis more accurate and procedures more efficient.

2.4 Concluding Remarks

This chapter reviewed the current landscape of technologies used for examining the uterine cavity, focusing mainly on hysteroscopic imaging and recent developments in 3D reconstruction tech-

niques. Traditional hysteroscopy is still the standard clinical tool, but since it only provides 2D images, it limits spatial understanding during the procedure. Stereoscopic systems do help with depth perception, but they are not commonly used in practice due to cost and hardware complexity.

Recent advances in endoscopic SLAM and deep learning-based approaches show strong potential for enabling real-time 3D reconstruction. However, their clinical translation is still limited by computational demands and the lack of annotated gynecological datasets.

To address these gaps, this work proposes a real-time depth-based reconstruction pipeline adapted for routine hysteroscopy. Built on the *EndoSLAM* framework, it prioritizes speed and accessibility while maintaining clinically useful spatial accuracy. In doing so, it aims to support more complete uterine evaluations and enable earlier detection of abnormalities during standard diagnostic procedures.

Chapter 3

Methodology

This work builds upon the *EndoSLAM* framework [26], which introduced a SLAM-based approach for monocular endoscopic depth and pose estimation. While the original system predicts both depth and camera trajectory in a self-supervised way (performing full simultaneous localization and mapping), this study focuses exclusively on the depth estimation component, adapting the *EndoSfMLearner* architecture [26] and using ground truth poses. By doing so, the proposed pipeline isolates and evaluates the performance of the depth prediction component, removing pose estimation errors as a confounding factor and allowing for a more accurate evaluation of depth map quality and 3D reconstruction outcomes.

Although this adjustment may seem to compromise real-time capabilities, it shifts attention toward analyzing the execution speed of both depth prediction and 3D reconstruction processes. The insights gained here are foundational for future re-integration into a fully real-time system, which is something that the current *EndoSLAM* pipeline does not yet achieve.

It is also important to note that the 3D reconstruction pipeline developed in this work is not part of the original *EndoSLAM* framework. It was independently implemented, using the predicted depth maps and external ground truth camera poses.

At a high level, the system used in this work can be summarized as follows:

- **Input Data** — RGB frames, ground truth depth maps, and GT camera poses;
- **Depth Prediction** — A modified DispResNet-based architecture is used for single-frame depth estimation, followed by conversion from disparity to depth.
- **3D Reconstruction** — The predicted depth maps are transformed into point clouds in world coordinates using the corresponding GT poses. These are then aggregated into a complete 3D model.

Furthermore, throughout this work, two datasets were used to support and validate the methodology: the original *EndoSLAM* dataset, and a newly acquired dataset captured from a physical 3D-printed uterine model that was purposely developed as part of this work. These are described in

the next section, followed by details about the depth prediction models and the 3D reconstruction process.

3.1 Datasets

This work relies on two datasets: the *EndoSLAM* dataset, which provides both real and synthetic endoscopic sequences, and a custom dataset created specifically for this project.

The *EndoSLAM* dataset offers a wide range of anatomical scenes and acquisition setups. Although it focuses on gastrointestinal organs rather than the uterus, it remains valuable for testing depth prediction and reconstruction methods. However, certain limitations, particularly in terms of data completeness and interpretability, motivated the creation of a second, fully controlled dataset. This new dataset features a physical uterus model and enables detailed evaluation under known geometric and visual conditions, with access to all acquisition characteristics such as camera intrinsics, depth scale, and resolution.

3.1.1 EndoSLAM Dataset

The *EndoSLAM* dataset includes a diverse set of sequences acquired using multiple camera systems — *HighCam*, *LowCam*, *MiroCam*, *PillCam*, *OlympusCam*, and *UnityCam* — and covering three gastrointestinal organs: the stomach, colon, and small intestine. These sequences comprise both real organ recordings (captured *ex vivo* from porcine specimens) and synthetic data generated in a virtual simulation environment [26].

In terms of camera systems and acquisition setups:

- *HighCam*, *LowCam*, *MiroCam*, and *PillCam* were used to record real *ex vivo* organ sequences. These are accompanied by precise Six Degrees of Freedom (6-DoF) ground truth pose data obtained via a robotic arm and, in many cases, high-fidelity 3D reconstructions generated from CT or structured light 3D scans.
- *UnityCam* was used to simulate endoscopic sequences in a photorealistic virtual environment. This synthetic dataset includes RGB frames, per-frame depth maps, and corresponding pose annotations.
- *OlympusCam* was used in a clinical context with a silicone colon phantom, as shown in the dataset's clinical validation section, and is also accompanied by CT-derived ground truth models.

Dataset Subset Selection: UnityCam (Stomach)

For this work, sequences focusing on the stomach were selected due to their morphological and textural similarities to the uterus, making them a more suitable surrogate for this application, as discussed in Section 2.2. Among these, the synthetic *UnityCam* subset proved particularly

useful, as it provides endoscopic-like views rather than open *ex vivo* organ recordings like the other cameras. Additionally, it includes both predicted depth maps and ground truth camera poses in a consistent and noise-free format.

Additionally, for the *UnityCam* synthetic dataset, the camera intrinsics matrix is provided and corresponds to the virtual camera used to generate the RGB images and also the GT depth maps (see Appendix C.3 for the intrinsic matrices). This matrix is essential for projecting the predicted or GT depth maps into 3D space and is used throughout the reconstruction pipeline.

However, it is important to note that the *UnityCam* subset does not provide a complete ground truth 3D reconstruction of the full scene. Moreover, inconsistencies were identified in the ground truth depth maps, which are discussed in more detail in Section 3.1.1.1.

Units and Data Format

Although the dataset documentation does not explicitly state the units used in the GT depth maps, several indicators strongly suggest that they are expressed in centimeters. This assumption is based on the original paper's evaluation of 3D surface reconstruction, where error thresholds are defined in centimeters (e.g., using ICP until an RMSE (Root Mean Squared Error) deviation of 0.001 cm is reached) [26]. Empirically, these depth maps exhibit depth values ranging roughly from 0 to 50, reinforcing this interpretation.

Furthermore, both the RGB images and the GT depth maps have a resolution of 320x320 pixels. The RGB images are stored as standard 8-bit per channel color images in PNG format, with filenames following a sequential naming convention (e.g., `image_0000.png`, `image_0001.png`, etc.). Each frame is associated with a CSV (Comma-Separated Values) entry containing the 6-DoF pose (translation and quaternion rotation) and an approximate timestamp. These GT camera poses are provided in meters, and the provided intrinsics matrix also reflects metric scale (i.e., measurements in meters), consistent with the virtual environment's geometry.

Analyzing the poses `.csv` file, each row corresponds to a frame in the sequence and follows the format:

```
tX, tY, tZ, rX, rY, rZ, rW, time(s)
```

Here, `tX, tY, tZ` represent the translation components of the pose, while `rX, rY, rZ, rW` describe the rotation in quaternion format. The final column is a timestamp in seconds.

Trajectory Visualization

Figure 3.1 illustrates the ground truth trajectory of the camera for the *UnityCam* stomach sequences. This plot helps visualize how the virtual camera moved through the environment, providing spatial context for the sequence of RGB and depth frames. The axis units are in meters, directly matching the translation values recorded in the pose file.

Beyond scale interpretation, this 3D view of the trajectory highlights the challenge of maintaining geometric consistency across frames and helps anticipate how depth predictions from different viewpoints must align to produce a coherent point cloud.

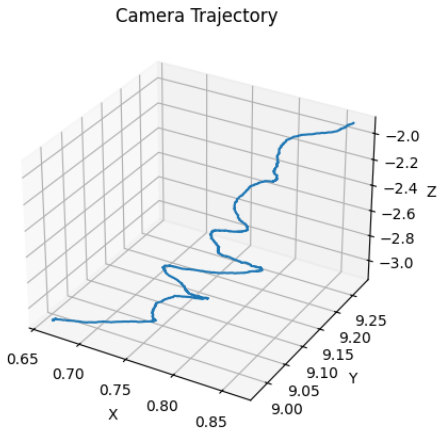


Figure 3.1: Example of the camera trajectory for the UnityCam stomach sequence, plotted using the ground truth pose data. The axes are shown in meters.

3.1.1.1 Limitations of the EndoSLAM UnityCam Subset

For the GT data, a significant limitation was identified in this subset. The depth ranges vary substantially across frames and exhibit inconsistent depth magnitudes and poor spatial coherence - for example, some frames display unrealistically large or small depth values that do not align with the expected stomach geometry in a simulated environment. This issue is clearly illustrated in Figure 3.2, which shows the minimum, maximum, and mean depth values across the sequence and highlights the erratic behavior of the GT depth data.

Due to this lack of robustness, even though each depth map can be individually evaluated using median scaling, the generated GT 3D model cannot be reliably used for quantitative comparisons or for meaningful evaluation of the predicted depth map reconstructions. As a result, it is not possible to properly validate the accuracy or fidelity of the predicted 3D geometry or the overall 3D reconstruction process. These issues are further discussed in Section 4.2.3, where a point cloud reconstruction and corresponding depth variation analysis are presented.

These limitations significantly affected the evaluation process and motivated the development of a new dataset, described in the following section, designed to provide better-controlled conditions for validating depth prediction and 3D reconstruction performance.

3.1.2 Uterus Phantom Dataset

While the *EndoSLAM* dataset offers valuable sequences, particularly from the synthetic *UnityCam* subset of the stomach, it presents limitations due to inconsistencies in the GT depth maps. Additionally, it lacks a reliable full-scene GT 3D reconstruction, and these issues complicate efforts to

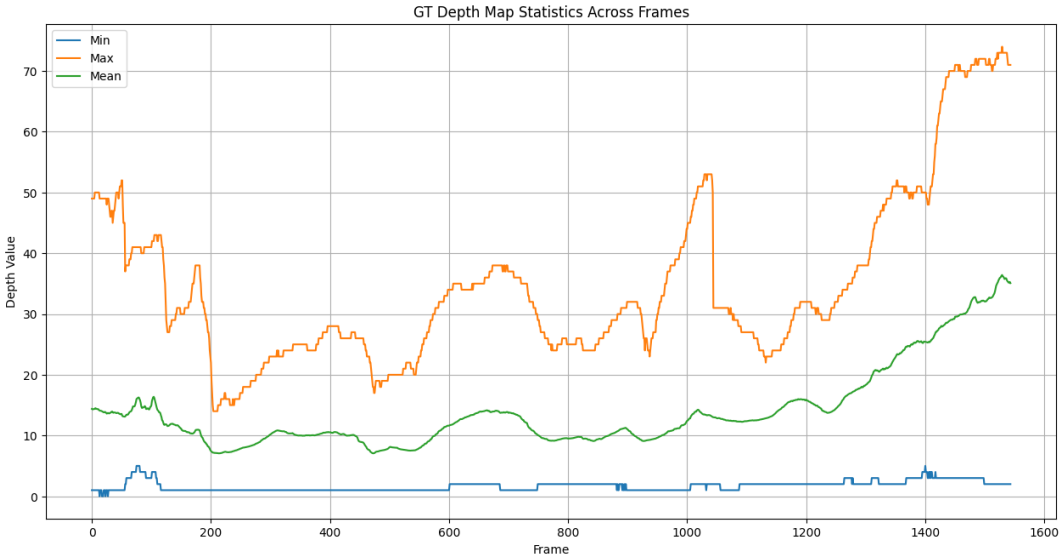


Figure 3.2: Per-frame statistics of the GT depth maps, including minimum, maximum, and mean depth values.

accurately assess the performance and reliability of the 3D reconstruction process.

To overcome these challenges, a custom dataset was developed using a uterus phantom. The motivation was not simply the absence of useful GT data, but the need for a dataset in which all acquisition parameters, such as geometry, materials, lighting, and camera properties, were fully known and controllable. This level of control was essential for isolating and accurately evaluating the performance of the depth prediction and 3D reconstruction pipeline under clearly defined conditions.

The dataset was constructed using a physical yet synthetic scene: a 3D-printed, anatomically inspired uterus model designed to resemble hysteroscopic conditions, particularly in terms of shape, texture, and coloration. Although it did not replicate biological fluids such as blood or mucus, the model provided a realistic structural representation. This approach enabled the creation of a controlled environment with accurate geometry, while also allowing a unique form of validation apart from the GT depth maps — the original 3D model used for printing could serve as a direct reference for assessing reconstruction fidelity.

Having that said, RGB images and depth maps were collected under controlled conditions, and ground truth camera poses were acquired using a robotic system. The following subsections describe the design of the physical model, the data acquisition process, and the structure of the resulting dataset.

3.1.2.1 Uterus Phantom Creation

3D Modeling

The anatomical model used in this study was created using Blender [4], an open-source 3D modeling software known for its user-friendly interface and extensive online support. As a starting point,

a publicly available mesh of the female reproductive system was obtained from Sketchfab [23], licensed under CC-BY-4.0. This mesh was then modified to isolate the uterus: the fallopian tubes, vaginal structures, and an abnormal internal mass were removed, preserving only the uterine body to better reflect a healthy target anatomical region. Both original and adapted models can be visualized in Figure 3.3.

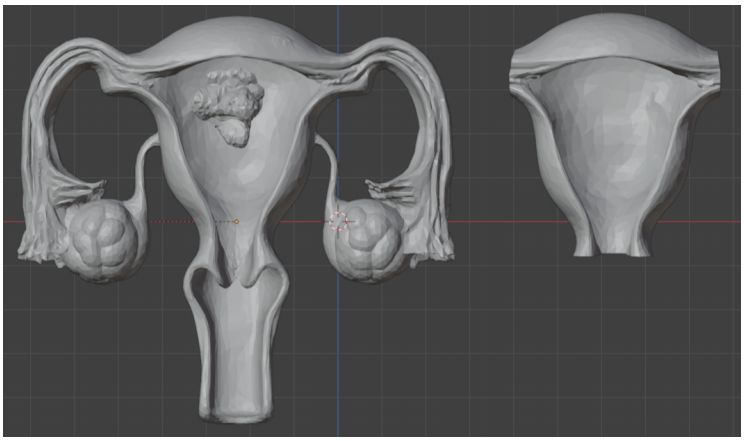


Figure 3.3: Comparison between the original reproductive system mesh (left) and the adapted uterus-only model (right) created in Blender.

Two variants of the uterus model were designed:

- One with a small frontal opening (Figure 3.3 on the right);
- Another with a coronal cut, offering a clear cross-sectional view (Figure 3.4).

Figure 3.4 shows a screenshot from Blender's viewport illustrating the coronal-cut uterus model from both front and side perspectives. Key physical dimensions, such as width, height, and depth, were annotated to guide mold sizing and ensure printability.

The annotated dimensions correspond to a final model size of approximately 0.1498 m in height, 0.1309 m in width (front view), 0.0162 m in thickness at the base (where the vaginal canal was removed), and 0.0650 m in depth. These measurements ensured anatomical plausibility while remaining compatible with the available 3D printer's build volume.

For each variation, a corresponding negative mold was also modeled to allow for silicone casting. However, after evaluating the complexity and print feasibility of both, the coronal-cut version was selected as the final design for fabrication. This version offered greater ease during both 3D printing and silicone casting, especially given the limitations of available equipment and materials, as explained in the following sections.

To complement the earlier model visualization, Figure 3.5 shows the negative mold designed in Blender for the coronal-cut uterus model. The mold consists of two halves: the front mold (left) and the back mold (right). To ensure proper alignment during the silicone casting process, six cylindrical keys were added — three on each side — allowing the mold parts to fit together precisely when assembled.

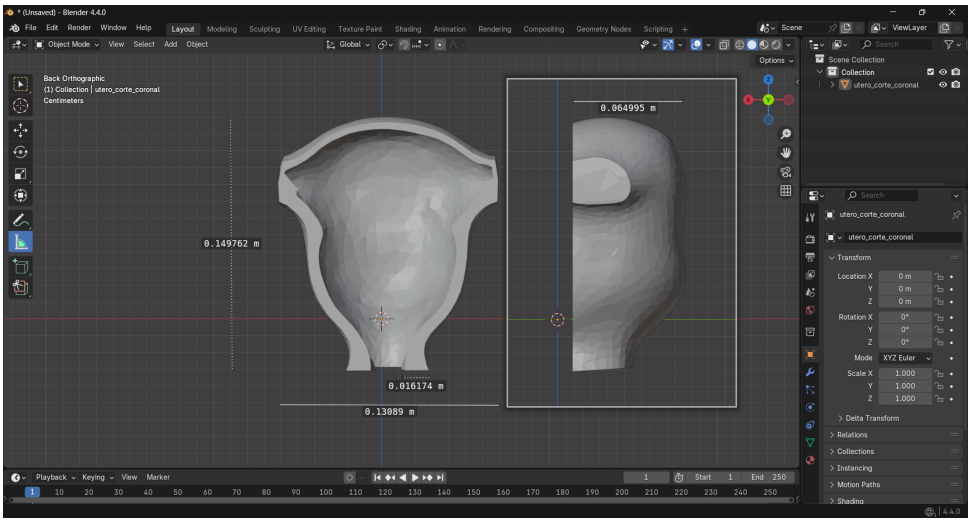


Figure 3.4: Blender viewport showing the final coronal-cut uterus model with annotated physical dimensions. The front view (top) displays height, width, and thickness; the side view (bottom) shows lateral depth.

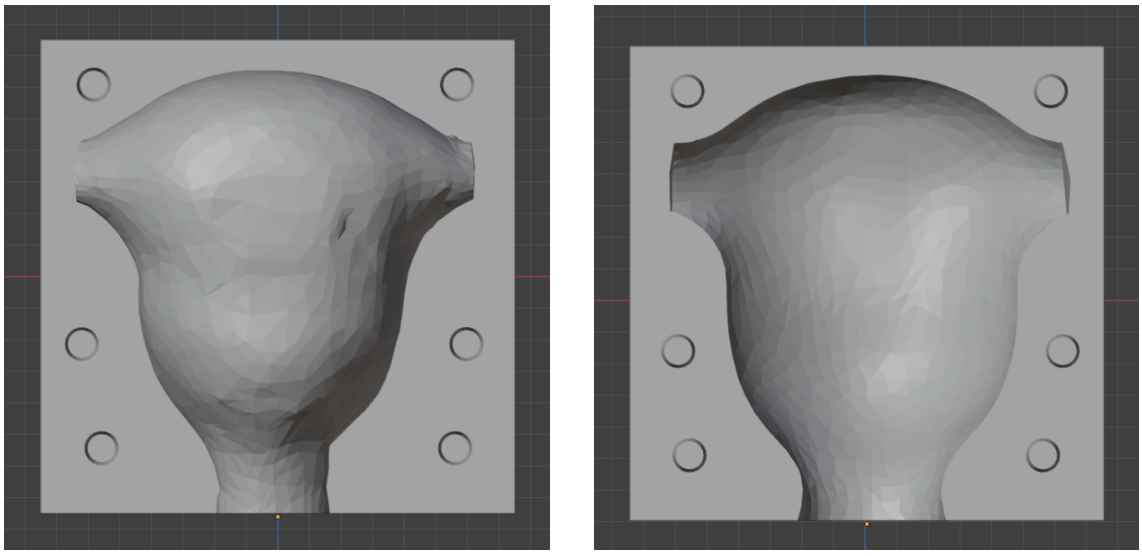


Figure 3.5: Blender view of the negative mold designed for the coronal-cut uterus model: front mold (left), back mold (right).

To better illustrate how the mold components align when assembled, Figure 3.6 shows additional screenshots from the Blender viewport rendered in wireframe mode. These views provide a clearer understanding of the complete model configuration, specifically, the correct positioning of the front and back mold halves, while also making the internal geometry and structural alignment more visible.

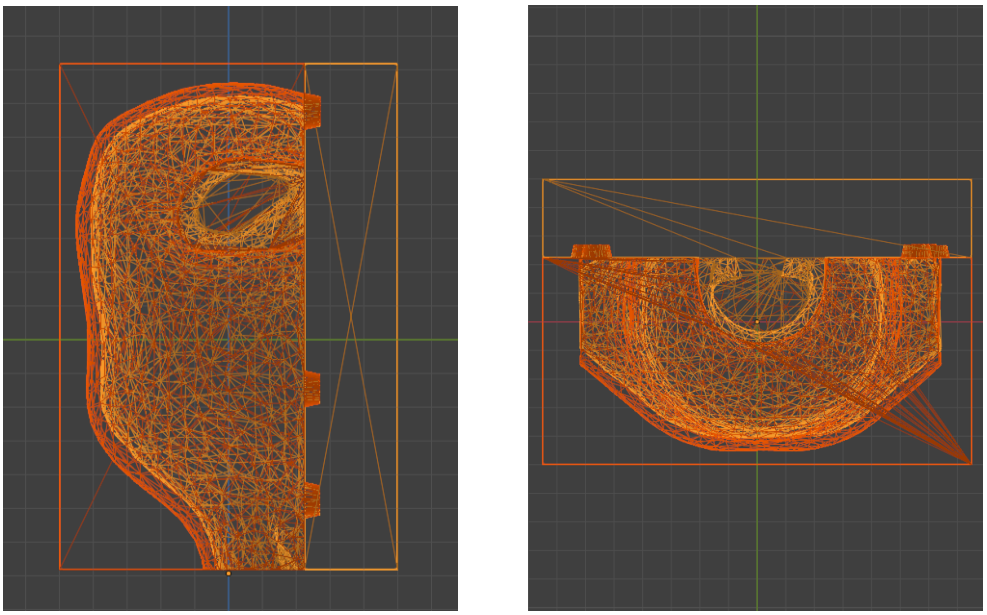


Figure 3.6: Wireframe view of the coronal-cut uterus mold in Blender: side view (left) and bottom view (right).

3D Printing

The final mold was exported from Blender and saved as an STL (Standard Tessellation Language) file for 3D printing. A standard FDM (Fused Deposition Modeling) 3D printer (Creality Ender 3) with PLA (Polylactic Acid, a biodegradable thermoplastic) filament was used, and these choices were primarily dictated by equipment constraints and material availability.

Although a resin printer for direct printing of the anatomical model would have been preferable, offering finer detail and smoother surfaces, the PLA mold approach provided sufficient strength and accessibility for its intended one-time use: casting the final model in silicone. The resulting printed mold is shown in Figure 3.7.

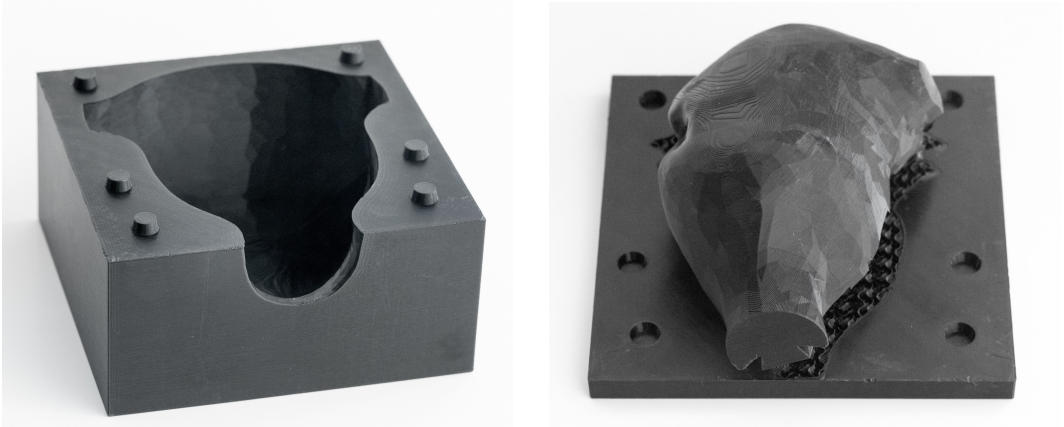


Figure 3.7: Photograph of the final 3D-printed PLA mold, printed in two halves. The remaining portion of the front part is shown on the left, and the back part on the right.

During the demolding process, some force was required to extract the silicone cast, which resulted in the front part of the mold breaking. As a consequence, no complete photograph of the mold's front half is available.

Silicone Casting

The PLA mold was filled with Ecoflex™ 00-50 silicone, a soft, skin-safe material that was pigmented with a "medium flesh" tone to resemble biological tissue. The pigmentation and softness were chosen to mimic biological tissue and provide realistic visual and physical conditions for data acquisition. After approximately three hours, the silicone had cured and the physical model could be successfully extracted from the mold. Figure 3.8 presents the resulting silicone uterus model.

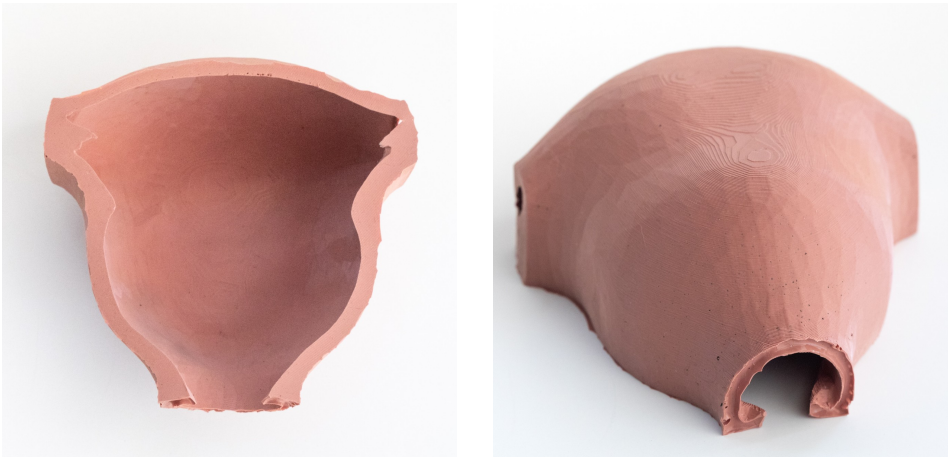


Figure 3.8: Final silicone uterus model captured after demolding. The anterior view is shown on the left, and the posterior view on the right.

3.1.2.2 Data Extraction

To capture RGB images, depth maps, and motion data from the silicone uterus model, an Intel® RealSense™ D435i camera was used [18]. This device features a stereo depth module, infrared sensors, and an inertial measurement unit (IMU), making it suitable for 3D reconstruction tasks that require both spatial and motion data.

The camera, along with its dimensions, is shown in Figure 3.9. Its physical size - 90 mm × 25 mm × 25 mm - posed a challenge in terms of scale compatibility with the relatively small uterine model (approximately 150 mm in height and 130 mm at its widest point). Nonetheless, it was selected due to its availability and capability to provide both RGB and depth data.

Figure 3.9: Intel® RealSense™ D435i camera (source: Intel Corporation [18]).

Camera Setup and Data Acquisition

Since the Intel RealSense D435i camera does not include a built-in pose sensor, a robotic arm was used to acquire precise ground truth camera poses. The camera was securely attached to the arm, enabling controlled movement and accurate tracking of its position throughout data collection.

The data acquisition took place in a shared laboratory environment (where the robotic arm was located), where it was not feasible to darken the room or employ a dedicated light source — conditions that could have improved both depth sensing performance and the quality of the RGB images used for depth prediction. As a result, data were acquired under ambient lighting provided by open windows and overhead ceiling lights.

Figure 3.10 presents the final setup. The setup includes the camera fixed on the robotic arm and the silicone uterus model positioned on the table. The model was placed inside the back mold for support due to its lack of structural rigidity.

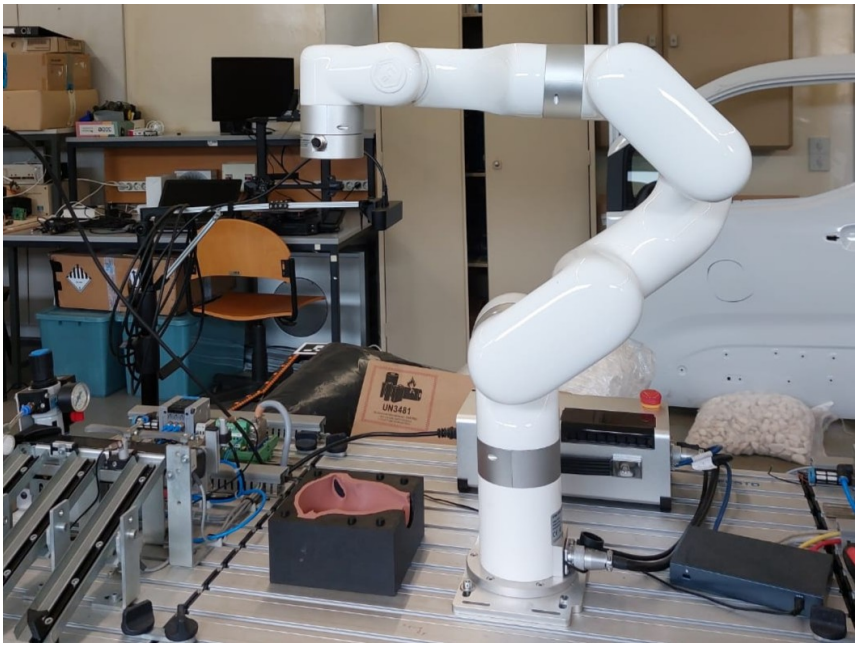


Figure 3.10: Laboratory setup showing the acquisition environment.

To improve robustness and generalization, the data acquisition procedure was repeated with the uterus model placed in various orientations relative to the camera's perspective:

- Front-facing;
- 90° rotations to the right and left;
- Rear-facing;
- Inclined positions in all four directions (front, back, right, left).

This diversity of viewpoints was intended to generate a more comprehensive depth map dataset and to support both per-frame evaluation and full 3D reconstructions for each individual orientation of the physical model.

It is important to note that data were captured at an approximate distance of 40 cm from the model, as the minimum functional range for the D435i in this configuration was 30 cm, and a small margin was added to ensure stable operation. While a closer range would have been preferable, given the small size of the model (approximately 15 cm), the available setup still allowed the full model to be captured, although some background elements were inevitably included in the frames.

Therefore, the following data were collected for each position:

- RGB images;
- Ground truth depth maps;
- IMU data, including linear acceleration and angular velocity;
- Ground truth camera poses, computed externally from the robotic arm's position, orientation, and velocity parameters.

All data streams (RGB images, depth maps, and IMU measurements), were recorded into `.bag` files using the Intel RealSense SDK (Software Development Kit). This format preserves synchronized sensor data and metadata in a compact binary format, allowing for later processing and extraction using Python libraries such as `pyrealsense2`.

Although the robotic system was not fully integrated and did not provide per-frame positional data or timestamps, the correspondence between poses and frames was established through interpolation based on known motion parameters, as detailed in the following subsection.

An example of the Intel RealSense Viewer interface is shown in Figure 3.11. It is divided into four viewports: the top left panel displays the live depth map (used as ground truth), while the bottom left shows the RGB stream. Notably, the RGB image appears closer than the depth map, due to the differing optical characteristics of the RGB and depth sensors. The top right and bottom right panels show the IMU gyro and accelerometer streams, respectively. Although IMU data was recorded, it was not directly used in this work.

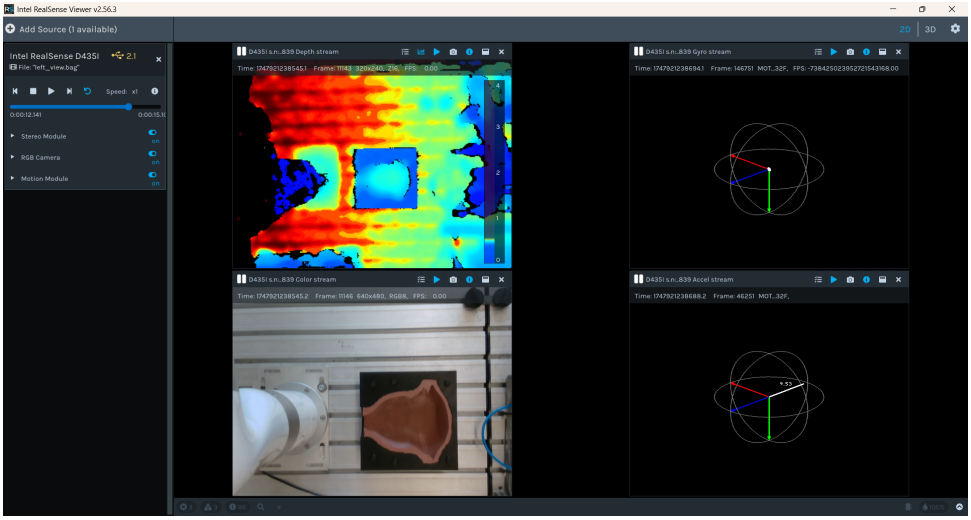


Figure 3.11: Screenshot of the Intel RealSense Viewer interface.

The RealSense SDK also provides separate intrinsic calibration parameters for both the RGB and depth sensors, which were extracted using Python and later used for depth alignment and 3D reconstruction tasks (see Appendix C.3 for the intrinsic matrices).

Units and Data Format

Similar to the *EndoSLAM* dataset, RGB images in the uterus phantom dataset are stored as PNG files with standard 8-bit per channel color encoding. Each image follows a sequential naming convention (e.g., `image_0000.png`, `image_0001.png`, etc.) and can be read using standard image libraries such as `PIL` or `OpenCV`. In this case, both the RGB images and the GT depth maps have a resolution of 480×640 pixels. Additionally, the GT depth maps are provided in millimeters and were converted to meters.

As for the GT camera poses, to maintain consistency with the *EndoSLAM* dataset, they were organized using the same CSV structure, where each row corresponds to a single frame and contains translation and rotation values. However, unlike *EndoSLAM* UnityCam subset, which uses quaternions, this dataset stores rotations as Euler angles (roll, pitch, yaw), originally recorded in degrees and later converted to radians. The translation values, initially in millimeters, were also converted to meters. These conversions, along with the one applied to the GT depth maps, were performed to ensure consistency with the metric scale used throughout the pipeline and are all applied later in the main 3D reconstruction script.

The resulting CSV structure follows the format:

```
tX, tY, tZ, rX, rY, rZ, time(s)
```

This format enabled straightforward reuse of existing parsing and transformation functions within the reconstruction pipeline, including the dataset class handlers (with minor adjustments) and functions such as `pose_vec2mat()`, which support both quaternion- and Euler-based rotation formats (provided in radians).

Frame-to-Pose Mapping

As previously explained, ground truth camera poses were derived from the known position, orientation, and velocity parameters of the robotic arm used in the acquisition setup. The data capture followed a linear, bidirectional motion path: the camera moved forward from an initial Y-position, paused for 5 seconds, and then returned along the same path.

Since the robotic arm did not output per-frame timestamps or positional data, the correspondence between poses and frames was computed through interpolation, using known parameters such as start and end positions, constant motion speed, frame count, and acquisition frame rate. The timestamps used were obtained directly from the Intel RealSense `.bag` file and were converted from milliseconds to seconds for consistency with the velocity units.

It is important to note that, at this stage, all pose values remained in their original units (millimeters for translation and degrees for rotation). The conversion to meters and radians was deferred to the dataset reading and pre-processing routines in the main 3D reconstruction script (see Appendix C.2, `prepare_gt_data_created_dataset`).

The expected number of frames for each motion segment was estimated based on known parameters: the total displacement along the Y-axis was 300 mm (from +150 mm to –150 mm), and the robotic arm was configured to move at a constant speed of 100 mm/s. This implied a motion duration of approximately 3 seconds in each direction. The RealSense device was operating at approximately 15 frames per second, leading to an estimated 45 frames per segment. However, after visual inspection, 48 frames were used for both forward and backward paths to ensure full coverage.

The static trajectory parameters were defined as follows:

- **Position (fixed):** `X = 200 mm`, `Z = 400 mm`
- **Orientation (Euler angles, degrees):** `Roll = 180`, `Pitch = 0`, `Yaw = 86`
- **Motion range:** `Y = 150 mm` to `Y = -150 mm`
- **Forward path:** 48 frames
- **Pause (static):** 79 frames
- **Backward path:** 48 frames

The Y-translation for each frame was calculated using linear interpolation. Let i denote the frame index within its motion segment. Then:

$$Y_i = \begin{cases} 150 - i \cdot \left(\frac{300}{N_f - 1}\right), & \text{for forward motion} \\ -150 + i \cdot \left(\frac{300}{N_b - 1}\right), & \text{for backward motion} \end{cases} \tag{3.1}$$

where:

- Y_i is the computed Y translation (in mm),

- i is the frame index within the current direction,
- N_f and N_b are the number of frames for the forward and backward motions, respectively (both set to 48).

To ensure alignment between poses and image frames, the first frame identifiers (IDs) for both depth and RGB streams were manually annotated from the `.bag` file. For instance, in the `front_view.bag` sequence, where the uterus model is oriented directly toward the camera, `first_depth_frame` and `first_rgb_frame` were set to 4974 and 4977, respectively.

Only the forward and backward motion segments were retained for pose and image alignment. From the 5-second pause interval, a single representative frame was kept to reduce redundancy and ensure consistency in reconstruction.

This logical structure ensured that each frame was paired with a corresponding interpolated pose and synchronized timestamp, producing a coherent camera trajectory suitable for subsequent depth prediction and 3D reconstruction, which can be visualized in Figure 3.12.

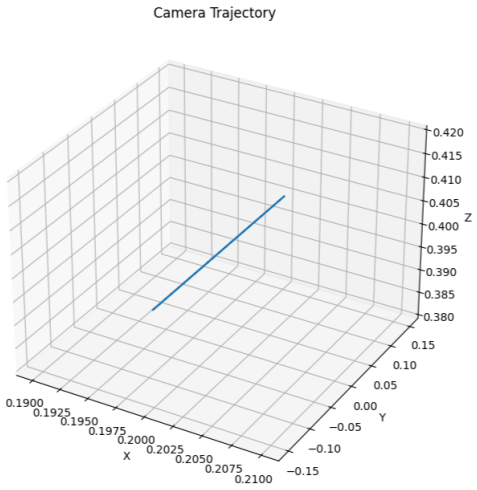


Figure 3.12: Camera trajectory for the uterus phantom, shown in meters.

A custom Python script was developed to perform this pose annotation, frame extraction, and data structuring. It handled the conversion of raw `.bag` recordings into organized folders containing RGB images, GT depth maps, and corresponding pose files. This ensured compatibility with the rest of the pipeline, and the full implementation is included in Appendix A.1.

RGB Frames Consideration

During data acquisition, the RGB images captured by the RealSense camera included substantial background elements from the laboratory environment. To ensure that the depth prediction model focused primarily on the anatomical region of interest, the silicone uterus phantom, a post-processing cropping step was applied to the recorded frames.

Based on the known fixed trajectory of the robotic arm, the consistent positioning of the phantom on the table, and visual inspection of the captured frames, an asymmetric horizontal crop

was defined to exclude irrelevant regions from the sides of each image. The crop fractions were empirically determined as follows:

- `crop_left_fraction = 0.43`
- `crop_right_fraction = 0.18`

These values specify the proportions of the original image width to be removed from the left and right sides, respectively. The cropping was then implemented in Python:

```
w_rgb = color_image.shape[1]
crop_left_rgb = int(w_rgb * crop_left_fraction)
crop_right_rgb = w_rgb - int(w_rgb * crop_right_fraction)
cropped_rgb = color_image[:, crop_left_rgb:crop_right_rgb, :]
```

Listing 3.1: Asymmetric cropping applied to RGB images.

This pre-processing step was essential for preparing the input to the monocular depth estimation model, ensuring that the network was trained and evaluated on the relevant content while reducing artifacts, background noise, and unrelated objects. A corresponding uncropping operation was later applied to the predicted depth maps, restoring them to their original resolution to maintain consistency with the intrinsic camera parameters.

Ground Truth Considerations

The ground truth captured in this dataset enables both frame-by-frame evaluation of predicted depth accuracy and full-scene comparison against the corresponding 3D reconstruction. This is made possible by the inclusion of high-quality depth maps, free from inconsistencies or unknown artifacts, ensuring data reliability and predictable outputs. As a result, the ground truth 3D model will not only provide a fair baseline for comparing the predicted 3D reconstructions, but may also serve as an independent reference for verifying the correctness of the overall reconstruction pipeline, given the known spatial configuration of the acquisition environment.

3.1.2.3 Limitations of the Uterus Phantom Dataset

Although the creation of a custom dataset brought significant advantages, such as full control over geometry, materials, and acquisition conditions, the setup was not without limitations.

One of the main challenges arose from the physical constraints of the recording setup. The silicone uterus model measured approximately 15 cm in length, but the Intel® RealSense™ D435i camera has a minimum functional sensing range of around 30 cm. This made it impossible to capture close-range images, which had originally been intended to better simulate endoscopic views. Instead, the camera recorded the entire scene, including the table surface and background elements, rather than tightly framing the anatomical model, reducing the overall visual realism.

In addition to its limited sensing range, the camera also lacked a built-in position tracking system, which would have been especially useful in this work, where pose estimation is not performed and the pipeline relies entirely on ground truth poses. To address this, a robotic arm was used to control the camera's motion and positioning. While this provided a stable and repeatable acquisition trajectory, it introduced further constraints, as it required operation within a fixed room, along a predefined path and viewpoint, reducing flexibility in scene setup and data acquisition.

Beyond that, printer availability was limited for the fabrication process, and only a standard FDM printer was accessible. While this was suitable for printing a mold to be filled with silicone, access to a resin printer would have been beneficial in terms of anatomical detail and fabrication ease. However, even with a higher-quality model, the intended results could not have been achieved without first addressing the more critical limitation posed by the camera.

Furthermore, it was not possible to acquire the data in a dark room with a single focused light source, which would have better replicated the illumination conditions typical of endoscopic procedures and could have improved the quality of the predicted depth maps.

Despite these constraints, the dataset remains valuable for assessing the generalization capabilities of monocular depth prediction models and for analyzing the 3D reconstruction process. It provides insights into performance under non-ideal yet realistic conditions and serves as a useful complement to fully synthetic or *ex vivo* datasets.

3.2 Depth Prediction Pipeline

Depth estimation is one of the core focuses of this project, forming the basis for generating a 3D understanding of the uterine cavity from monocular hysteroscopic video. The challenge lies in predicting dense, per-pixel depth using only a single RGB image, in real time and under constrained clinical hardware. To accomplish this, a CNN-based depth estimation model was employed as the core of the system.

The adopted architecture follows an encoder–decoder design, where a convolutional neural network (CNN) processes the input image and outputs a disparity map (which is converted to depth) with per-pixel predictions. The architecture was based on the *EndoSLAM* framework [26], specifically on its EndoSfMLearner depth network, originally developed for endoscopic video frames. Although different approaches were explored, the original structure was essentially maintained, with the only modification aimed at improving the inference speed of depth predictions. An overview of the depth prediction pipeline is illustrated in Figure 3.13.

Figure 3.13: Overview of the depth prediction pipeline.

3.2.1 EndoSLAM Depth Prediction Module

In *EndoSLAM* article, the authors proposed a depth estimation model based on CNNs using an encoder–decoder architecture, named EndoSfMLearner [26]. The objective of this module is to predict depth from monocular endoscopic images or video using a self-supervised learning strategy. Its structure is inspired by the Disparity Network (DispNet) architecture [**?**], and a visual overview is presented in Figure 3.14.

```
[RGB Image]
    |
[ResNet Encoder] --> Conv1/2 --> Residual Blocks (x4) --> Feature
   Pyramid
    |
[Depth Decoder]
  |-- Upsample Block 1 (with skip from encoder)
  |-- Upsample Block 2
  |-- Upsample Block 3
  \-- Final Disparity Head
    |
[Disparity Map] --> (1/x) --> [Depth Map]
```

Listing 3.2: Architecture of the depth estimation module (simplified ASCII layout).

The **encoder** is responsible for extracting relevant visual features from the input image. It starts with a 7×7 convolution layer (with 64 filters and stride 2), followed by a Rectified Linear Unit (ReLU) activation and a max pooling operation. The ReLU introduces non-linearity and helps the model learn complex patterns, while max pooling reduces the spatial resolution and keeps only the most prominent activations, helping make the network more robust and efficient. This structure is based on the well-known Residual Network (ResNet) architecture [14], which introduced residual connections that help avoid vanishing gradients and improve training efficiency for deeper networks.

Following this, the encoder passes through four residual blocks (RB64, RB128, RB256, RB512) that capture increasingly abstract features across multiple scales. These blocks use skip connections, which help gradients flow more easily during training, mitigate the vanishing gradient problem, and preserve spatial detail.

To support training stability and faster convergence, the encoder uses ImageNet-pretrained weights [8]. This means it was initially trained on a large-scale natural image dataset, which helps the model generalize better to medical images, as low-level features like edges, contours, and textures are often transferable across domains.

On the other hand, the **decoder** reconstructs the disparity map from the encoded features through progressive upsampling. It consists of five convolutional blocks (Ce256 to Ce16), each including convolutional layers followed by Exponential Linear Unit (ELU) activations. Similarly to ReLU,

ELU promotes sparsity while allowing small negative values to stabilize training. The decoder also incorporates skip connections from the encoder, which help preserve spatial details that would otherwise be lost due to downsampling. As a result, it is able to recover spatial resolution and generate a dense disparity map, the inverse of depth.

The final layer is a convolution followed by a sigmoid activation (referred to as Cs16), which outputs disparity values between 0 and 1. These predicted disparities are then inverted (via $1/x$) to obtain the final depth map.

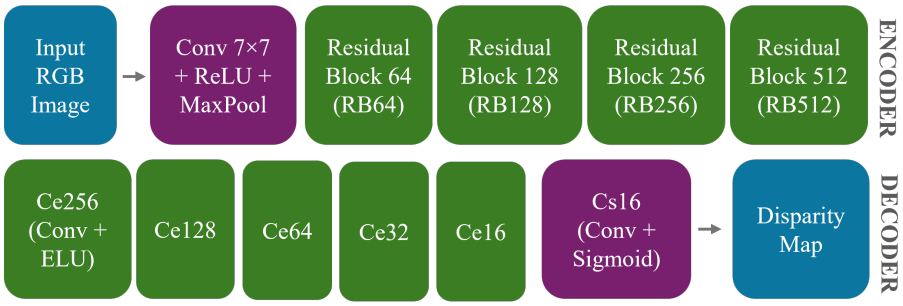


Figure 3.14: Detailed architecture of the EndoSLAM depth estimation module.

Lastly, the **inference** process used to generate depth predictions from input images is based on the script `test_disp.py`, which loads the pretrained model (with either `ResNet-18` or `ResNet-50` as the backbone), sets it to evaluation mode, and processes input images accordingly.

The model relies on pretrained weights from the file `pretrained_models.pth`, included in the official *EndoSLAM* GitHub repository [12]. These weights were originally trained on a combination of synthetic and *ex vivo* endoscopic data, and are reused here to produce new predictions.

Each input image is resized to the expected resolution, normalized using ImageNet statistics (mean = 0.45, std = 0.225), and passed through the model to generate a disparity map. The resulting output is then inverted ($1/x$) to obtain the final depth map, which is saved as a 3D NumPy array (`predictions.npy`) for later use in reconstruction.

3.2.2 Considered and Tested Approaches

The primary objective of this work was to improve the inference speed of depth predictions. Although potential improvements in output quality were obviously desirable, the original *EndoSLAM* predictions were already sufficiently accurate for endoscopic image reconstruction, particularly for the UnityCam subset (simulated stomach environment). As such, quality was not the main driver behind the approaches considered.

With that in mind, several alternatives to the default *EndoSLAM* depth estimation module were considered. These fall into three main categories:

- **Partial architecture modifications:** either replacing the encoder (e.g., with MobileNet or EfficientNet) or the decoder (e.g., with U-Net);

- **Complete architecture replacement:** testing fully independent models such as MiDaS and FastDepth, which provide their own pretrained weights and inference pipelines;
- **Inference-level optimizations:** exploring strategies like quantization, pruning, input downscaling, batch processing, `ONNX` export, and the use of `TensorRT` acceleration.

It is also worth noting that the model uses a `ResNet-18` backbone, a relatively lightweight and efficient architecture that already contributes to acceptable inference times.

Architecture Approaches

The motivation behind substituting parts of the original depth prediction architecture was to improve inference speed, by evaluating lightweight and efficient models such as MobileNet [15] and EfficientNet [35], which are known for their fast runtime performance and compact design. However, these modifications introduced several practical constraints.

First, partial replacements of the architecture (e.g., changing only the encoder or decoder) proved to be unfeasible due to structural incompatibilities. In particular, using a new encoder with the existing decoder would require the output feature maps to match in both spatial dimensions and channel depth. Most lightweight encoders produce features with different shapes, making them incompatible without also modifying the decoder. Conversely, replacing only the decoder would require it to adapt to the feature map structure of the original encoder, which again would necessitate architectural redesign or full retraining.

Moreover, one of the key advantages of the original *EndoSLAM* pipeline was the use of pretrained weights, and that became inaccessible when modifying only part of the network. Since the encoder and decoder are tightly coupled, altering one component invalidates the pretrained weights for the entire model, thereby requiring extensive fine-tuning or retraining. Given the lack of paired RGB and GT depth data, these options were not feasible within the scope of this work.

As a result, only complete models that provided their own pretrained weights could be realistically evaluated. FastDepth [43], although designed as a lightweight model for real-time depth estimation, could not be successfully integrated. Either pretrained weights were unavailable or incompatible, or the inference results were completely unsatisfactory. Consequently, FastDepth was excluded from further testing.

Among the models considered, only MiDaS [27] was successfully implemented and tested. It is a complete model with publicly available pretrained weights and is known for strong generalization across datasets, making it an interesting option to evaluate the performance. Its results and limitations are discussed in Section 3.2.2.

Inference Approaches

From the considered strategies, batch processing and input downscaling were the most effective in improving inference time and were therefore included in the final implementation. These were

complemented by additional implementation-level enhancements, such as OpenCV-based image handling and optional mixed precision support, which are discussed in more detail in Section 3.2.3.

By contrast, dynamic quantization, pruning, and model export to `ONNX` format were tested but ultimately discarded, as they produced negligible performance gains or were outperformed by the selected optimization techniques.

Acceleration frameworks such as `TensorRT` [44] were also considered but could not be tested due to the lack of a compatible CUDA-enabled GPU on the available hardware. Nonetheless, they remain a promising direction for future optimization if suitable infrastructure becomes available.

MiDaS Approach

As an alternative to the main architecture based on the *EndoSLAM* pipeline, this work tested the MiDaS model (Mix Depth and Scale), a neural network designed to estimate depth maps from a single RGB image [27]. One of the key advantages of MiDaS is its strong generalization capability, as it performs well across diverse environments and datasets without requiring retraining. This made it a particularly interesting option to test, since it could be used without needing to fine-tune or adapt the model specifically to endoscopic data.

Furthermore, MiDaS is a pretrained model that leverages large-scale image datasets such as *ImageNet* [8] to learn robust visual representations. Its architecture follows an encoder–decoder structure, similar to that of *EndoSLAM*, but with some notable differences:

- **Encoder:** responsible for extracting high-level features from the input image. Different backbone architectures are available, including ResNet, EfficientNet, and Vision Transformer (ViT), depending on the variant used. These encoders leverage pretrained weights to improve performance and reduce the need for training—especially important when domain-specific data is limited.

- **Decoder:** a dense convolutional structure that performs progressive upsampling and multi-scale feature fusion to generate a coherent and spatially accurate depth map.

To support inference, the input image underwent several pre-processing steps within the MiDaS codebase, including:

- Color space conversion and normalization;

- (Optional) contrast enhancement using CLAHE (Contrast Limited Adaptive Histogram Equalization).

Once pre-processed, the image is passed through the MiDaS model, which outputs a depth map. It is important to note that MiDaS produces relative depth maps, meaning the output values reflect only the relative distance between objects in the scene rather than absolute physical units.

The following MiDaS variants were tested in this work: `MiDaS`, `MiDaS_small`, `DPT_Large`, and `DPT_Hybrid`. Additionally, the `DPT_SwinV2_L_384` was attempted but failed to run successfully, either due to incompatibility with the available pretrained weights or GPU memory limitations on the computer used.

It is important to highlight that all tests were done on the *EndoSLAM* UnityCam stomach dataset. This dataset was chosen because it closely resembles the visual and structural characteristics of real endoscopic environments, making it more appropriate for assessing model suitability for clinical use.

In particular, CLAHE pre-processing was enabled for the `DPT_Hybrid` variant, where it was observed to enhance local contrast and produce more consistent depth outputs. Other pre-processing techniques, such as histogram equalization and gamma correction, were briefly evaluated but did not yield significant improvements, and were not adopted for any of the tested variants.

All of the successfully tested models produced valid depth maps. While the larger DPT-based variants offered higher visual quality, they were significantly slower and more memory-intensive. Moreover, all outputs still lacked the level of detail and sharpness observed in the original *EndoSLAM* predictions, making them less suitable for the specific requirements of this application.

3.2.3 Final Depth Prediction Module

Before implementing any functional changes, the original structure was refactored to improve modularity, readability, and maintainability. The depth estimation pipeline was reorganized into separate scripts for the encoder, decoder, pre-processing utilities, and pretrained model weights. In addition, an `__init__.py` file was introduced to combine the encoder and decoder into a unified model interface, effectively instantiating the `DispResNet` architecture (see Appendix B for the implementation). This restructuring helped isolate the inference logic from the model definition, making subsequent experimentation and testing more efficient. The updated directory structure is shown in Listing 3.3.

```
updated_depth_predictions/
|- inference.py # Main prediction pipeline
|- models/
| |- __init__.py # Model interface
| |- encoder.py # ResNet encoder
| \- decoder.py # Depth decoder
|- pretrained/
| \- pretrained_dispnet.pth # Pretrained model
\- utils/
  \- transforms.py # Image pre-processing
```

Listing 3.3: Structure of the updated depth prediction code (simplified ASCII layout).

The decision to retain the original *EndoSLAM* model was based on the comparative limitations of alternative approaches, summarized in Table 3.1. While other models were considered, most were either incompatible with the pretrained weights or failed to deliver improved performance under project constraints.

Table 3.1: Comparison of depth estimation alternatives considered.

Approach	Advantages	Limitations	Status
EndoSLAM (original)	Good accuracy on UnityCam; pretrained weights available	Slightly slower than desired	Retained
MiDaS	Strong generalization; ready-to-use weights	Lower sharpness; slower inference	Discarded
FastDepth	Lightweight; designed for speed	No usable pretrained weights; poor performance	Discarded
U-Net decoder	Potential for better spatial detail	Incompatible with encoder weights; needs retraining	Not tested
MobileNet / EfficientNet encoder	Fast and compact	Decoder needs adaptation; pretrained weights unusable	Not tested

Inference Optimization Strategy

Given the decision to maintain the original ResNet-18-based depth estimation model from *EndoSLAM*, the focus shifted solely to improving inference speed and performance without compromising output quality.

To support this, the original `test_disp.py` inference script was used as a base to develop a customized version, `inference.py`, whose full implementation is provided in Appendix B, Listing B.2. The most effective strategies tested and adopted included:

- **OpenCV-based pre-processing:** faster and more efficient image loading, resizing, and normalization using OpenCV rather than PIL or PyTorch utilities;
- **Input downscaling:** resizing the input images to 75% of their original resolution provided a good trade-off between speed and prediction quality;
- **Batch processing:** significantly reduces inference time. Batch sizes of 1, 2, 4, 8 and 16 were tested, with 8 being optimal on the available hardware.

In addition, the output format was also revised: all generated depth maps are now saved in a single `.npy` file, which facilitates batch analysis and downstream evaluation.

Lastly, mixed precision support was integrated into the script as an added feature. Although it was not actively used or tested on the available machine, the pipeline is capable of leveraging compatible GPU hardware and drivers when available. This feature can provide additional performance improvements and is therefore worth enabling when supported.

To evaluate and optimize inference performance, the pipeline was instrumented to record detailed runtime statistics. These included total prediction time, model inference time, and pre-processing time. From these, key performance metrics such as average per-image times and frame rate (FPS) were derived. The FPS was computed as the inverse of the average time per image (i.e., FPS $= 1/\text{time}$), based either on total prediction time or inference time alone, depending on the context. These measurements provided insight into computational efficiency and helped assess the system's potential for real-time clinical deployment.

For a more thorough analysis of the effect of batch size on performance, inference experiments were conducted with varying batch configurations. The resulting data were statistically analyzed to guide final parameter selection; full details of this analysis are presented in Section 4.1.1.

3.3 3D Reconstruction Pipeline

The final stage of this work involves the generation of a 3D model from predicted or ground truth depth maps. This is achieved by first projecting image-space depth values into 3D points in the camera coordinate frame using the known camera intrinsics, and then transforming these points into a common world coordinate frame using the corresponding camera poses. By aggregating the transformed point clouds across the full trajectory, a spatially consistent 3D reconstruction of the anatomical scene is obtained. This model can be visualized as a point cloud or further processed into a surface mesh [7].

An overview of this reconstruction process is illustrated in Figure 3.15.

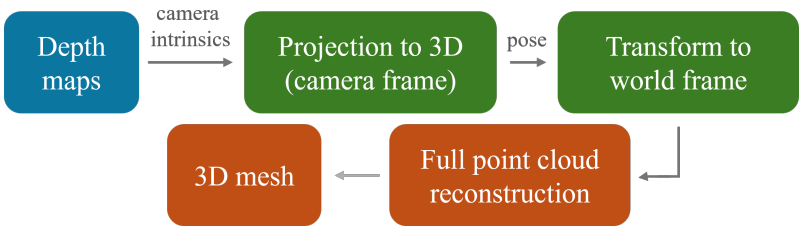


Figure 3.15: Overview of the 3D reconstruction pipeline.

The goal of this stage was not only to produce an accurate 3D model but also to enable reconstruction at or near real-time speeds. Furthermore, the 3D reconstruction pipeline used in this work was independently implemented, with the exception of three transformation-related functions that were adapted from the original *EndoSLAM* codebase. The rest of the *EndoSLAM* reconstruction framework was only used to understand certain dataset-related parameters and to better interpret the required pre-processing steps.

3.3.1 Data Loading and Reading

To load and structure the RGB frames, depth maps, and pose data used in the 3D reconstruction process, two dataset handler classes were implemented: one for the *EndoSLAM* UnityCam stomach dataset and another for the custom-phantom dataset. Although both share a similar logic, their formats differ slightly—most notably in unit conventions.

The uterus phantom dataset applies conversions directly within the handler classes: translation values are converted from millimeters to meters, and Euler angles from degrees to radians. In contrast, the *EndoSLAM* UnityCam data is used as-is.

Both implementations follow an object-oriented design and are included in Appendix C, Section C.1, Listings C.2 and C.3. Each dataset class includes readers for RGB frames, depth maps, and poses, and exposes a unified access interface. A useful utility method, `get_all_depth_maps()`, is also included for batch loading of ground truth depth data.

Example usage is shown below:

```
dataset = Dataset_unity_cam(path_frames, path_depth_maps, path_poses)
depth_map_predictions = np.load(path_depth_predictions, mmap_mode='r')

original_img = dataset[0]["Image"]
pose = dataset[0]["Pose"]
gt_depth_map = dataset[0]["Depth_map"]
predicted_depth_map = depth_map_predictions[0]
gt_depth_maps = dataset.get_all_depth_maps()
```

Listing 3.4: Example usage of the dataset class and depth prediction loading.

3.3.2 Depth Maps

Before depth maps can be converted into 3D point clouds, several pre-processing steps are applied to ensure geometric consistency, spatial alignment, and fair evaluation. These steps vary slightly depending on the dataset and the type of depth map (ground truth vs. predicted). The following operations were applied:

For the *EndoSLAM* dataset, all pose values are already provided in meter - the target unit for reconstruction, as discussed in Section 3.1.1. However, the GT and predicted depth maps differ in both units and format: GT depth maps are in centimeters, while predicted maps are either already in meters. Additionally, they differ in resolution: the GT maps are 320×320, while the predicted outputs are typically 256×832. Considering these differences, the overall pre-processing steps applied were:

- **GT Depth Maps**: The values are inverted (`gt_depth_map.max() - gt_depth_map`) and then converted from centimeters to meters (by dividing by 100);

- **Predicted Depth Maps:** First, the maps are resized to match the resolution of the original RGB frames, ensuring consistency with the camera intrinsics. Then, for single-frame comparisons only, median scaling is applied to align their scale with that of the GT maps.

In addition, a circular mask is applied to both depth maps. For the predicted maps, it removes irrelevant border regions introduced by the endoscopic view. For the GT maps, this step is done to ensure fair and consistent comparison by aligning the effective field of view with that of the predictions.

It is important to clarify that `prepare_data()` is the function used for processing individual frames, such as for depth map comparison or single-frame point cloud generation, and includes the median scaling step. Median scaling is commonly applied in monocular depth estimation when predicted depths are in a relative scale, allowing alignment with a reference (e.g., ground truth). In this work, median scaling becomes particularly critical due to the inconsistencies in GT depth values, as discussed in Section 3.1.1.1. Without this correction, quantitative comparisons would be unreliable.

Conversely, for full-sequence 3D reconstruction, median scaling is not performed. Instead, the predicted depth maps are just resized to match the original RGB resolution to preserve spatial coherence with the intrinsic matrix, which is handled separately by the `prepare_predicted_data()` function. This alternative pre-processing pipeline reflects practical constraints: in real clinical applications, GT depth maps are often unavailable, making median scaling infeasible and unsuitable.

Finally, for the purposes of depth map visualization and evaluation, the pre-processing applied to the predicted maps for single-frame comparison (i.e., including median scaling) should be considered. Based on that, the following additional steps were applied:

- **Depth Map Visualization**: Both GT and predicted depth maps are normalized for visualization (before the mask application);
- **Depth Map Comparison**: No additional pre-processing steps are applied. The pre-processed depth maps (after resizing, scaling, and masking) are used directly for metric evaluation.

For the uterus phantom dataset, all pose values and GT depth maps are defined in millimeters, while the predicted depth maps appear to be in meters. The GT depth maps have a resolution of 480×640, whereas the predicted maps are cropped, typically to around 480×384. As a result, they require different pre-processing steps:

- **GT Depth Maps**: The maps are first converted to meters by dividing by 1000. Since the camera was positioned approximately 500 mm above the table surface, any depth value greater than 600 mm is discarded (set to zero) to isolate the anatomical model and nearby objects. Although content beyond 500 mm is known to be irrelevant, a threshold of 600 mm was used to provide a margin of safety and avoid accidentally discarding valid regions.
- **Predicted Depth Maps (cropped)**: These maps are uncropped using to restore their original dimensions, matching the full RGB resolution and, consequently, the corresponding

intrinsics matrix. The cropped-out regions are filled with zero values (black), preserving the spatial structure. This step is necessary due to the initial cropping applied to the RGB images, as discussed in Section 3.1.2.2.

Finally, for the purposes of depth map visualization, both maps are normalized, as done for the *EndoSLAM* dataset. However, these depth maps are not directly suitable for quantitative evaluation at this stage, since the predicted depth maps are generated from the RGB camera and the GT maps are captured by a different sensor with a wider field of view (as described in Section 3.1.2.2). Therefore, the two are not spatially aligned in the image plane and, consequently, direct pixel-wise comparison is not meaningful. Only after both maps are projected into the global 3D coordinate system, using their respective intrinsics and camera poses, can a fair and geometrically consistent comparison be performed.

These pre-processing routines are implemented in the functions `prepare_data()`, `prepare_predicted_data()`, and `prepare_data_created_dataset()`, along with auxiliary utilities such as `apply_circ_mask()`, `resize_median_scaling()` and `uncropping_predicted_depth()`. Each function is adapted to the specific structure and requirements of the corresponding dataset. Together, they ensure consistent input formatting for point cloud generation, mesh reconstruction, and depth map evaluation. Full implementations are provided in Appendix C.2.

To quantitatively assess the quality of the predicted depth maps in the *EndoSLAM* dataset, the Mean Absolute Error (MAE), Mean Squared Error (MSE), and Root Mean Square Error (RMSE) were computed, which measure absolute and squared differences between the predicted and ground truth depth values. Additionally, two perceptual similarity metrics were used: Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index Measure (SSIM), which assess the overall visual fidelity and structural coherence of the predicted depth maps relative to the ground truth. All comparisons were performed using depth maps that had been resized, median-scaled, and masked.

- Mean Absolute Error (MAE):

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}\left|D_i^{\text{GT}} - D_i^{\text{pred}}\right| \tag{3.2}$$

- Mean Squared Error (MSE):

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(D_i^{\text{GT}} - D_i^{\text{pred}})^2 \tag{3.3}$$

- Root Mean Square Error (RMSE):

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(D_i^{\text{GT}} - D_i^{\text{pred}})^2} \tag{3.4}$$

- Peak Signal-to-Noise Ratio (PSNR):

$$\text{PSNR} = 20 \cdot \log_{10} \left(\frac{\text{MAX}_{D^{\text{GT}}}}{\sqrt{\text{MSE}}} \right) \tag{3.5}$$

- Structural Similarity Index Measure (SSIM): The SSIM index is computed using local means, variances, and covariances between the predicted and ground truth maps, defined as:

$$\text{SSIM}(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{3.6}$$

  where μ and σ denote the local means and variances, and C_1, C_2 are small constants for numerical stability.

3.3.3 Point Clouds Generation

After the depth maps are pre-processed, the next step is to convert them into point clouds.

Point Clouds in the Camera Frame

For each frame in the dataset, a depth map $D(u,v)$ is available, either predicted or from ground truth. Using the known camera intrinsics K, each depth value is back-projected into a 3D point in the camera coordinate system using the standard pinhole camera model [7]:

$$\mathbf{P}^{\text{camera}} = D(u,v) \cdot K^{-1} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{3.7}$$

where (u,v) denotes the pixel coordinates in the image plane, $D(u,v)$ is the corresponding depth value, K is the camera intrinsic matrix, and $\mathbf{P}^{\text{camera}}$ is the resulting 3D point in the camera coordinate frame.

This equation is derived from the perspective projection model, which maps 3D scene geometry to the 2D image plane. Its inverse enables 3D reconstruction from depth values [7, Equations 13.2, 13.4–13.5].

The matrix K encodes the camera's internal parameters, namely, the focal lengths (f_x, f_y) and the principal point (u_0, v_0), and is structured as:

$$K = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

In this work, different intrinsic matrices were used depending on the dataset and the source of the depth map:

- For the *EndoSLAM* UnityCam dataset, a single intrinsic matrix is used for both ground truth and predicted depth maps. This matrix corresponds to the RGB camera used to render the synthetic data and is applied consistently, as all predictions are made from RGB input;
- For the uterus phantom dataset, two distinct intrinsic matrices are used:
  - One for the RGB camera, used to project the predicted depth maps;
  - One for the depth sensor, used to project the ground truth maps captured by the RealSense depth module.

Each intrinsic matrix is selected based on the source of the depth map to maintain geometric consistency in the resulting point clouds (see Appendix C.3 for the full set of matrices). Moreover, the back-projection from image to camera coordinates is performed by the `pixel2cam()` function, which applies Equation 3.7 using the inverse intrinsic matrix (see Appendix C.4). Importantly, this function excludes pixels with zero depth values, which typically correspond to invalid or missing measurements.

After obtaining the 3D points in the local camera frame, additional post-processing is applied depending on the dataset:

EndoSLAM Dataset:

- Depth Thresholding: A percentile-based filter (`apply_depth_threshold()`, set to the 75^{th} percentile) is applied to remove extreme depth values. This reflects the typical structure of endoscopic imagery, where the camera progresses along a tubular anatomical path and the scene often ends at a close tissue wall or surface;
- Axis Reordering: The axes of the resulting 3D points are reordered (`cam_coords = cam_coords[:, [0, 2, 1]]`) to match Unity's coordinate system, ensuring consistency with OpenCV-based operations elsewhere in the pipeline.

Uterus Phantom Dataset:

- No Depth Thresholding: Since the anatomical model sits on a table and the full scene is visible, no threshold is applied. Background surfaces like the table may be useful for contextual information;
- No Axis Reordering: The camera follows OpenCV conventions by default, so the output 3D points are already aligned correctly.

After these steps, the resulting point cloud represents the local geometry of the scene in the camera coordinate frame.

Point Clouds in the World Frame

Each point cloud is transformed from the local camera coordinate system to a global world coordinate frame using the known 6-DoF camera pose at the time of capture. These poses are provided either as:

- translation vectors and Euler angles, or
- translation vectors and quaternions.

To convert these into 4×4 homogeneous transformation matrices, the `pose_vec2mat()` function is used. This function internally calls either `euler2mat()` or `quat2mat()` to construct the appropriate rotation matrix, depending on the rotation representation. The transformation is then applied as:

$$\mathbf{P}^{\text{world}} = T_i \cdot \begin{bmatrix} \mathbf{P}^{\text{camera}} \\ 1 \end{bmatrix} \tag{3.9}$$

where $T_i \in \mathrm{SE}(3)$ is the camera-to-world transformation matrix at frame i, and $\mathbf{P}^{\text{camera}}$ is the 3D point in camera coordinates.

This operation follows the standard rigid-body transformation in SE(3), as described in [7, Section 13.3]. By accumulating transformed point clouds from multiple frames, a spatially consistent global 3D model is generated.

The functions `pose_vec2mat()`, `euler2mat()`, and `quat2mat()` were adapted from the original *EndoSLAM* codebase [12], and integrated into the reconstruction pipeline (see Appendix C.4).

At this stage, the single-frame point clouds from both datasets are ready for visualization and registration. However, for direct comparison, only those from the *EndoSLAM* dataset are immediately suitable. In the case of the custom uterus phantom dataset, an additional post-processing step is required to restrict the comparison to the relevant region of interest.

Although both point clouds are already expressed in the same world coordinate frame, the GT depth maps cover a much wider field of view than the predicted ones, which are limited to the narrower region captured by the RGB camera and further cropped to focus on the area of interest. As a result, the GT point clouds include background geometry that is not present in the predicted point clouds.

To enable a fair comparison, the GT point clouds are cropped to match the spatial region covered by the predicted ones. This is done by determining the minimum and maximum x and y coordinates from the predicted point cloud and applying those bounds to crop the GT point cloud in the world coordinate frame. The z-axis is left unchanged, as it represents depth. This ensures that both clouds describe the same physical area, specifically, the region of the table where the uterus phantom was placed, allowing for meaningful evaluation. Notably, this cropping step can only be performed after both point clouds have been projected into the world coordinate system.

3.3.4 Full Point Cloud Reconstruction

Once the per-frame point clouds are computed and transformed into the global world coordinate frame, they can be aggregated across the entire sequence to form the final 3D model. This is achieved by looping through all frames, generating each point cloud, and concatenating them, since they are already aligned in the same reference frame.

As a post-processing step for visualization, voxel grid downsampling can optionally be applied to the generated point clouds. This is performed using the `voxel_down_sample(voxel_size)` function, which reduces the density of the point cloud by aggregating points within uniform voxel cells. This step can be applied to either individual single-frame point clouds or to the final 3D reconstruction, and is especially useful for making the model easier to interpret and improving rendering performance.

As before, the full 3D point cloud generated from the GT depth maps of the uterus phantom dataset must be cropped to match the spatial extent of the predicted reconstruction, in order to ensure both visual alignment and a fair basis for metric comparison.

To achieve this, a post-processing step can be applied directly to the final world-frame point clouds. Specifically, an axis-aligned bounding box was computed from the predicted point cloud using the `predicted_pcd.get_axis_aligned_bounding_box()` function. This bounding box (`bbox`) was then used to crop the ground truth point cloud:

```
1 bbox = predicted_pcd.get_axis_aligned_bounding_box()
2 gt_pcd_cropped = gt_pcd.crop(bbox)
```

Listing 3.5: Cropping ground truth point cloud using the predicted bounding box.

To quantitatively compare the full 3D reconstructions, as well as individual point clouds, the Chamfer Distance and Hausdorff Distance were computed. These metrics evaluate geometric similarity between the predicted and ground truth point clouds by measuring spatial discrepancies between corresponding points, and are particularly suitable for comparisons when the point clouds differ in density or contain a different number of points.

3.3.5 3D Mesh Reconstruction (Optional)

Moreover, to produce a continuous surface representation from the aggregated point cloud, an optional mesh reconstruction step can be performed. This converts the discrete point cloud into a watertight surface using Poisson surface reconstruction, which infers shape continuity from local geometry.

The process consists of three main steps:

1. Normal Estimation: Surface normals are estimated for each point using a KD-tree neighborhood search;

2. Poisson Reconstruction: A mesh is generated using Open3D's screened Poisson reconstruction algorithm, with configurable parameters such as reconstruction depth and point density threshold;

3. Post-Processing: Low-density vertices are filtered out, and the resulting mesh is cleaned by removing degenerate triangles, duplicate vertices, and non-manifold edges.

This full routine is encapsulated in the `create_mesh()` function. Although the mesh offers a visually smoother and continuous representation of the surface, it is considered optional in this pipeline and, in practice, it did not significantly improve the evaluation or interpretation of results in either of the datasets used.

For evaluation, the reconstructed mesh can also be compared to its source point cloud using geometric metrics such as the Hausdorff Distance. However, meshes were not used to compare predicted against ground truth reconstructions, as point clouds were sufficient for this purpose.

3.4 System Overview

Implementation Environment

All components of the proposed pipeline were implemented in Python, using the *Visual Studio Code* development environment. Due to system limitations, all experiments were executed on CPU. The implementation relied on PyTorch, with supporting libraries including NumPy, OpenCV, and Matplotlib.

The codebase is designed to be portable and reproducible on any system with Python and the required dependencies.

Pipeline Workflow

The envisioned workflow processes hysteroscopic video frames sequentially, treating each image as a monocular input for depth estimation. While every frame can be processed in theory, selective frame skipping may be employed to maintain real-time responsiveness without significantly compromising reconstruction quality. Over time, the depth predictions from individual frames are aggregated to generate a cumulative 3D point cloud representation of the uterine cavity.

This point cloud may serve as the basis for generating a full 3D mesh, offering clinicians the ability to zoom into specific areas, review previously inspected regions, or assess spatial relationships with greater precision. Such capabilities are particularly valuable in identifying subtle abnormalities or ensuring full cavity coverage, goals that are often difficult to achieve using 2D visualization alone. However, implementation details such as a graphical interface or interactive navigation will be left for future work.

Nevertheless, the program is intended to run on a standard computer, requiring only a functioning Python environment. All components are designed to be lightweight and portable, allowing for straightforward execution without the need for specialized hardware.

Chapter 4

Results and Discussion

4.1 Depth Prediction Results

4.1.1 Inference Optimization

To improve the execution speed of the system without altering the depth prediction model itself, an updated inference pipeline, `inference.py`, was implemented. The original architecture from the *EndoSLAM* framework, including both the encoder and decoder, was preserved. This decision was based on the fact that the existing model already offered a good balance between speed and accuracy, especially when compared to alternative approaches.

Therefore, the focus was placed on accelerating the inference stage while maintaining depth map quality at a level acceptable for practical use. The following optimizations were applied:

- **OpenCV-based pre-processing:** Faster image loading and formatting;
- **Input downscaling:** Reduced computational load during inference;
- **Batch processing:** Enabled simultaneous processing of multiple frames.

Mixed precision inference was also implemented as an optional feature. However, since the experiments were performed on a CPU-only setup without CUDA support, it did not provide any noticeable speed improvement.

Note: While the term "inference" is correctly used here to describe the full prediction pipeline (from input image to final depth output), it also has a more specific meaning in deep learning, that is referring only to the forward pass through the depth estimation model (implemented here as *DispResNet*, which combines the encoder and decoder). In this section, both uses appear: "inference pipeline" refers to the overall process, whereas "inference time per image" refers specifically to the time taken by the model's forward pass.

Having said that, performance measurements were collected to evaluate the impact of these changes, including total execution time (with breakdowns of inference and pre-processing), average per-frame inference time, and resulting frame rate (FPS). A detailed analysis of the results

obtained with each implementation, along with the rationale behind the parameter choices, is presented in the following subsections. All evaluations and runtime measurements were conducted using the *EndoSLAM* UnityCam stomach dataset, comprising 1,548 images.

To establish a meaningful benchmark and enable comparison with the proposed improvements, the evaluation begins by measuring the inference speed of the original *EndoSLAM* implementation. In this baseline setup, depth prediction was performed using the `test_disp.py` script. Under standard conditions, the system achieved the following performance on the UnityCam stomach dataset:

```
Inference Time (per image):  0.145 seconds
Frame Rate:  6.90 FPS
```

A more detailed breakdown from a complete run is presented below:

- Total prediction time (full pipeline): 298.14 seconds (approx. 4 minutes and 58 seconds);
- Total inference time: 224.49 seconds;
- Total pre-processing time: 69.24 seconds (average of 0.045 seconds per image).

The total prediction time includes pre-processing, inference, and post-processing. However, the post-processing step is minimal - involving only the conversion of disparity maps to depth maps and saving the results - a process consistent across both the original and updated pipelines, and thus not considered significant enough to require separate analysis.

OpenCV-Based Pre-processing

The "OpenCV-based pre-processing" refers to replacing the original image loading and formatting pipeline with an OpenCV-based implementation. While this modification does not affect the model's inference time directly, it significantly reduces the time spent on pre-processing, thereby improving the overall prediction time.

In this evaluation, no additional optimizations, such as batch processing or input downscaling, were applied yet. The only changes involved restructuring the code and implementing OpenCV-based image handling. Therefore, inference was performed with no batch and using full-resolution input images.

Since the impact of this optimization is primarily on image loading and preparation, the change is best understood by comparing the complete execution time, including both pre-processing and inference. Table 4.1 presents the results, comparing the updated approach against the original *EndoSLAM* pipeline.

The most notable improvement was observed in the pre-processing stage, where the OpenCV-based implementation reduced the average pre-processing time per image from 0.045 to 0.005 seconds. This corresponds to an approximate reduction of 0.04 seconds per image, or about 62 seconds over the 1,548-frame sequence. Given that this optimization specifically targeted pre-processing, the result aligns with expectations.

Table 4.1: Impact of OpenCV-Based Pre-processing Compared to Original Pipeline

Metric	**Original EndoSLAM**	**Optimized Pipeline (OpenCV)**
Total prediction time (s)	298.14	**230.17**
Pre-processing time (s)	69.24	**7.15**
Inference time (s)	224.49	221.85
Per-Image Pre-processing Time (s)	0.045	**0.005**
Per-Image Inference Time (s)	0.145	0.143

In conclusion, the OpenCV-based pre-processing contributed to a faster and more modular prediction pipeline and was therefore retained for all subsequent experiments.

Downscaling

With the codebase already organized and OpenCV-based pre-processing implemented, input downscaling was introduced as the next optimization. To assess its impact, inference was performed with a 75% input resolution (downscaling), while keeping all other conditions unchanged.

The choice of 75% downscaling was based on empirical testing of multiple configurations. Initially, a 50% input resolution was tested, which resulted in a significant speedup but caused a noticeable drop in depth map quality. The resolution was then gradually increased in steps (e.g., 60%, 75%, 90%) to find a balance between speed and quality. Among these, a 75% resolution emerged as an effective middle ground, providing a meaningful reduction in inference time while preserving acceptable prediction quality.

To qualitatively assess the time impact of input downscaling, the results were compared against the previous configuration using full-resolution input. Table 4.2 provides a breakdown of execution time and each component's contribution to the total improvement, followed by per-image performance metrics in Table 4.3.

Table 4.2: Execution Time Breakdown and Improvement Contribution from Input Downscaling

Component	**Full Resolution**	**75% Downscaled**	**% of Total Improvement**
Total Prediction Time (s)	230.17	**147.77**	**35.8%** (82.4 s)
Pre-processing Time (s)	7.15	**5.64**	1.2% (1.5 s)
Inference Time (s)	221.85	**140.05**	35.3% (81.8 s)

Table 4.3: Effect of Input Downscaling on Per-Image Performance

Metric	**Full Resolution**	**75% Downscaled**	**Improvement**
Per-Image Pre-processing Time (s)	0.005	**0.004**	20.0% faster
Per-Image Inference Time (s)	0.143	**0.090**	37.1% faster

The main benefit of input downscaling was expected to occur during inference, as reducing image resolution directly decreases the computational workload during the model's forward pass. This expectation was confirmed: the implemented downscaling led to a substantial improvement in efficiency, reducing the average inference time per image from 0.143 to 0.090 seconds. This corresponds to a 37.1% reduction in per-image inference time, while maintaining comparable depth prediction quality, as will be demonstrated later. In practical terms, this also increased the frame rate from 6.99 to 11.11 FPS.

Nevertheless, a more modest improvement of approximately 0.001 seconds per image was observed in pre-processing time. While the absolute difference is small, it represents a 20% reduction relative to the original pre-processing time. This is consistent with the implementation logic, as downscaling occurs early in the pre-processing pipeline, allowing subsequent formatting steps to operate on smaller data and complete more quickly.

Overall, this optimization resulted in a 35.8% reduction in total prediction time for the 1,548-frame sequence, comprised of a 35.3% improvement from reduced inference time and a 1.2% contribution from pre-processing. This confirmed its practical value, justifying the 75% downscaling step inclusion in all later stages of the pipeline.

Batch Processing

Building upon the organized inference pipeline, already incorporating OpenCV-based pre-processing and 75% input downscaling, batch processing was introduced as the next optimization. To evaluate its impact on inference performance, the system was tested with all other components held constant while varying the batch size.

The model was executed with batch sizes of 1, 2, 4, 8, and 16. For each run, key performance metrics were recorded, including total prediction time, per-frame inference time, and the resulting frame rate.

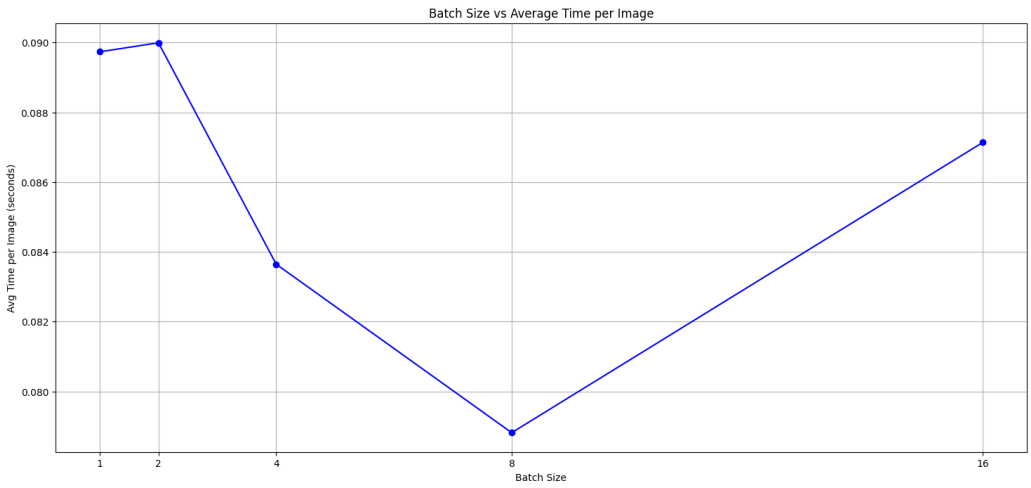


Figure 4.1: Effect of batch size on inference time per image.

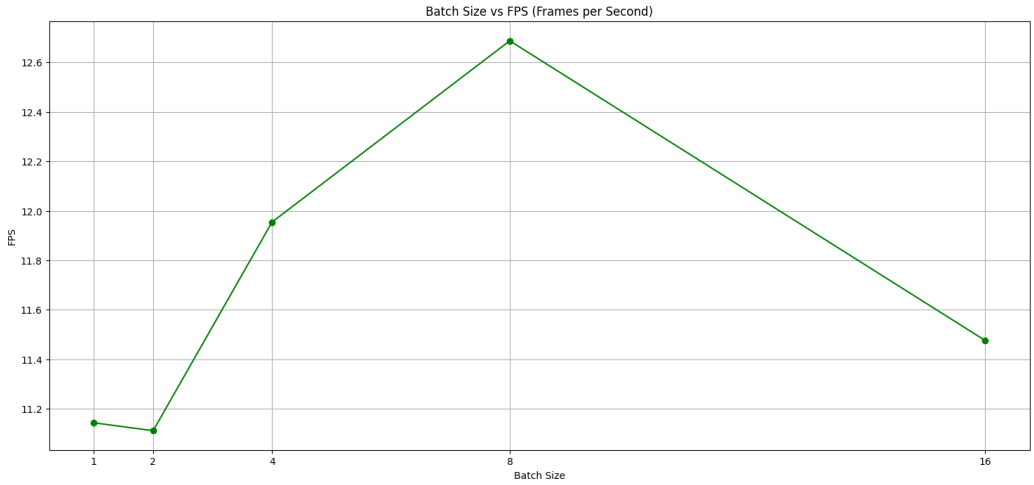


Figure 4.2: Effect of batch size on inference speed (FPS).

The results, shown in Figure 4.1 and Figure 4.2, illustrate the relationship between batch size and both the inference time per image and frame rate. As expected, increasing the batch size generally led to improved performance, up to a point. The best result was achieved with batch size 8, which yielded the lowest average inference time per image, approximately 0.0788 seconds. Beyond this point, performance began to degrade slightly, likely due to hardware limitations and memory overhead.

To further validate these results, a statistical analysis was conducted using the per-frame inference time values collected from multiple runs at each batch size.

First, to assess whether the observed differences across batch sizes were statistically significant, a non-parametric analysis was performed. A Shapiro–Wilk test was applied to each batch size group to evaluate the normality of the inference time distributions, and the results indicated that the data deviated significantly from a normal distribution in all cases ($p < 0.001$). Additionally, Levene's test revealed unequal variances among groups, violating the assumptions required for parametric tests such as one-way ANOVA.

As a result, the Kruskal–Wallis H-test was employed as a non-parametric alternative to determine whether at least one group differed significantly. The test yielded a highly significant result, confirming differences among batch sizes:

Kruskal–Wallis H-statistic: 2079.31, $p < 0.0001$

Following this, Dunn's post hoc test with Bonferroni correction was applied to perform pairwise comparisons using a significance threshold of $p < 0.05$. The resulting p-values are summarized in Table 4.4. Additionally, Figure 4.3 visually illustrates these results, showing inference time per image grouped by batch size, with significance labels (letters) indicating statistically similar or distinct groups. In this plot, circles represent outliers, and triangles indicate the mean values for each batch size.

Table 4.4: Dunn's Test Pairwise p-values Between Batch Sizes

Batch Size	**1**	**2**	**4**	**8**	**16**
1	1.000	1.000	4.03×10^{-73}	2.50×10^{-306}	0.0226
2		1.000	7.74×10^{-68}	2.23×10^{-295}	0.1756
4			1.000	1.17×10^{-81}	6.47×10^{-51}
8				1.000	1.26×10^{-258}
16					1.000

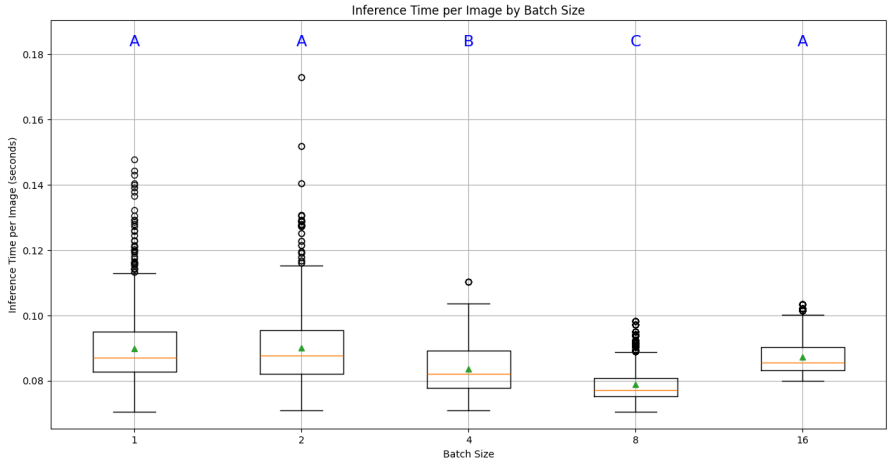


Figure 4.3: Inference time per image grouped by batch size. Statistically distinct groups (Dunn's test, Bonferroni-corrected) are labeled above each box.

These results confirm that the performance gains observed with batch size 8 are statistically significant and not due to random variation. While batch size 4 also showed significant improvements over smaller configurations (e.g., batch sizes 1 and 2), batch size 8 outperformed all others both in statistical significance and actual per-frame inference time. For these reasons, it was ultimately selected as the final configuration for the pipeline.

Furthermore, an inference run using batch size 8 was conducted to validate the overall effectiveness of batch processing. Compared to the previous pipeline stage (which included OpenCV-based pre-processing and 75% input downscaling but no batching), the following improvements were observed:

Table 4.5: Execution Time Breakdown and Improvement Contribution from Batch Processing

Component	**No Batching**	**Batch Size 8**	**% to Total Improvement**
Total Prediction Time (s)	147.77	**129.12**	**12.6%** (18.65 s)
Pre-processing Time (s)	5.64	**5.36**	1.5% (0.28 s)
Inference Time (s)	140.05	**122.15**	12.1% (17.90 s)

Table 4.6: Effect of Batch Processing on Per-Image Performance

Metric	**No Batching**	**Batch Size 8**	**Improvement**
Pre-processing Time per Image (s)	0.004	**0.003**	25.0% faster
Inference Time per Image (s)	0.090	**0.079**	12.2% faster

Batch processing was expected to improve overall runtime efficiency by enabling the system to leverage data-level parallelism and reduce per-sample overhead during inference. This is particularly relevant in deep learning inference, where processing multiple inputs simultaneously can amortize fixed computational costs over the batch.

This expectation was confirmed, since with a batch size of 8, the per-image inference time dropped from 0.090 to 0.079 seconds, and the total prediction time for the full 1,548-frame sequence was reduced by approximately 12.7%. Although the pre-processing time improved only modestly, the frame rate increased from 11.11 to 12.73 FPS, demonstrating a clear gain in processing throughput and efficiency.

Mixed Precision

Although mixed precision was implemented as part of the updated inference pipeline, it did not lead to a noticeable improvement in execution time. This is consistent with expectations, as the system used for evaluation lacked GPU support with proper CUDA acceleration, which is typically required to benefit from mixed precision operations.

To confirm this, performance was tested under identical conditions with and without mixed precision enabled (using all previously selected optimizations). Although it had already been verified that CUDA support was not available on the system - by checking for GPU availability in Python - the test was conducted for completeness. As expected, the inference time per image was nearly identical in both cases: 0.080 seconds without mixed precision and 0.079 seconds with it. This minor difference is likely attributable to runtime variability rather than any actual performance gain.

Nonetheless, mixed precision support was retained in the code as an optional feature for environments equipped with CUDA-enabled GPUs, where it may produce meaningful speedups.

Summary of Optimized Inference Performance

The final optimized inference pipeline included the following improvements:

- Organized codebase (modularized structure and timing logic);
- OpenCV-based image pre-processing;
- 75% input downscaling;
- Batch processing with batch size 8;

- Optional support for mixed precision (not evaluated due to lack of CUDA support).

To evaluate the impact of the full set of optimizations implemented in the pipeline (excluding mixed precision), performance was compared against the original pipeline from the *EndoSLAM* framework. A summary of the resulting improvements is presented in Tables 4.7 and 4.8.

Table 4.7: Execution Time Breakdown: Original vs. Final Optimized Inference Pipeline

Component	**Original EndoSLAM**	**Optimized Inference**	**Total Contribution**
Total Prediction Time (s)	298.14	**129.12**	**56.7%** (169.02 s)
Pre-processing Time (s)	69.24	**5.36**	21.4% (63.88 s)
Inference Time (s)	224.49	**122.15**	34.3% (102.34 s)

Table 4.8: Per-Image Performance Comparison

Metric	**Original EndoSLAM**	**Optimized Inference**	**Improvement**
Total Time per Image (s)	0.193	**0.083**	56.7% faster
Pre-processing Time per Image (s)	0.045	**0.003**	93.3% faster
Inference Time per Image (s)	0.145	**0.079**	45.5% faster

In conclusion, the optimized pipeline significantly improved performance across all stages. The total prediction time was reduced by approximately 56.7% (from 298.14 s to 129.12 s), corresponding to a savings of over 2 minutes and 49 seconds for the full 1,548-frame sequence. A substantial portion of this gain resulted from improvements in pre-processing, primarily due to OpenCV-based image handling and input downscaling, which together accounted for 21.4% of the total speedup. Inference efficiency also improved by 34.3%, largely driven by the use of lower-resolution inputs and the introduction of batch processing.

At the per-image level, pre-processing time dropped by 93.3% (from 0.045 s to 0.003 s), while inference time decreased by 45.5% from its original value. As a result, the overall system throughput nearly doubled, increasing from 5.19 FPS to 11.98 FPS, improving the feasibility of near real-time operation for practical, large-scale applications.

Figure 4.4 provides a breakdown of the execution time across different components of the pipeline (total, inference, and pre-processing) for each optimization stage. This visualization highlights how individual steps of the system were affected by each performance improvement. Finally, to support one of the most reflective performance indicators, the cumulative impact of each optimization step is illustrated by the FPS gains shown in the bar plot of Figure 4.5.

4.1.2 Depth Map Quality Evaluation

Although the core model architecture remained unchanged throughout the optimization process, several modifications were introduced in the inference pipeline—most notably the OpenCV-based

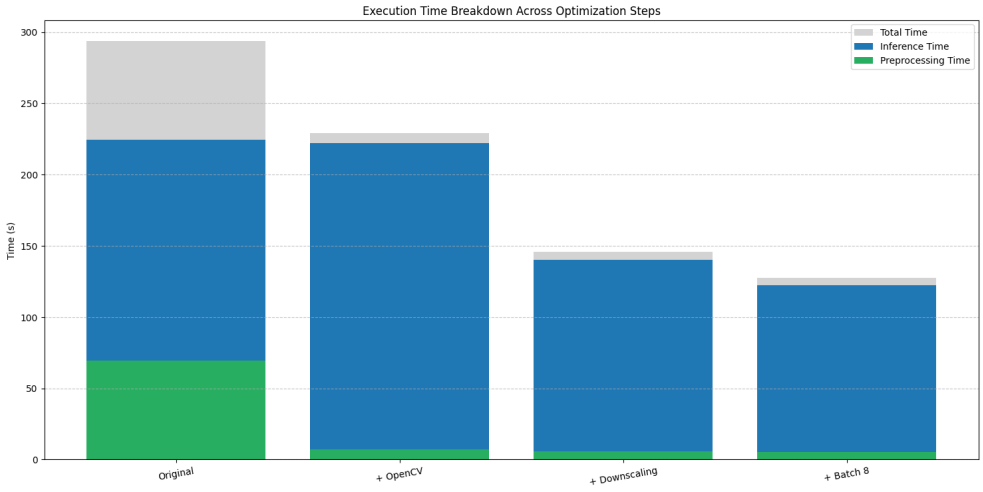


Figure 4.4: Execution time breakdown across optimization steps.

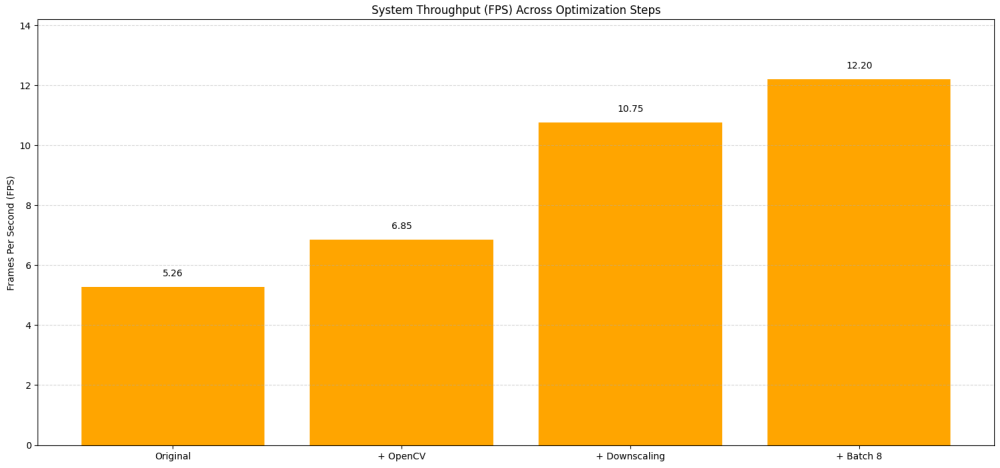


Figure 4.5: System throughput (in frames per second) across optimization steps.

pre-processing and 75% input downscaling. To ensure that these optimizations did not significantly degrade the quality of the depth predictions, both qualitative and quantitative comparisons were conducted between the outputs from the original `test_disp.py` script and those from the newly developed `inference.py` pipeline.

To assess whether the optimized pipeline preserved prediction accuracy, the depth maps generated by the updated pipeline were compared against those produced by the original *EndoSLAM* implementation using standard error metrics:

- Mean Absolute Error (MAE): 0.0446
- Mean Squared Error (MSE): 0.0039
- Root Mean Squared Error (RMSE): 0.0606

These values are closely aligned with those reported in the original *EndoSLAM* article, indicating that the applied optimizations did not result in any significant loss of prediction quality.

To further support this conclusion, Figure 4.6 presents a visual comparison of depth maps predicted using both pipelines for three representative frames: the 1st, 700th, and 1400th in the sequence. While minor visual differences are present, they are not substantial and do not affect the interpretation of the scene. Crucially, the regions with greater depth remain consistent across versions, suggesting that the optimized inference procedure preserved the perceptual quality of the predictions.

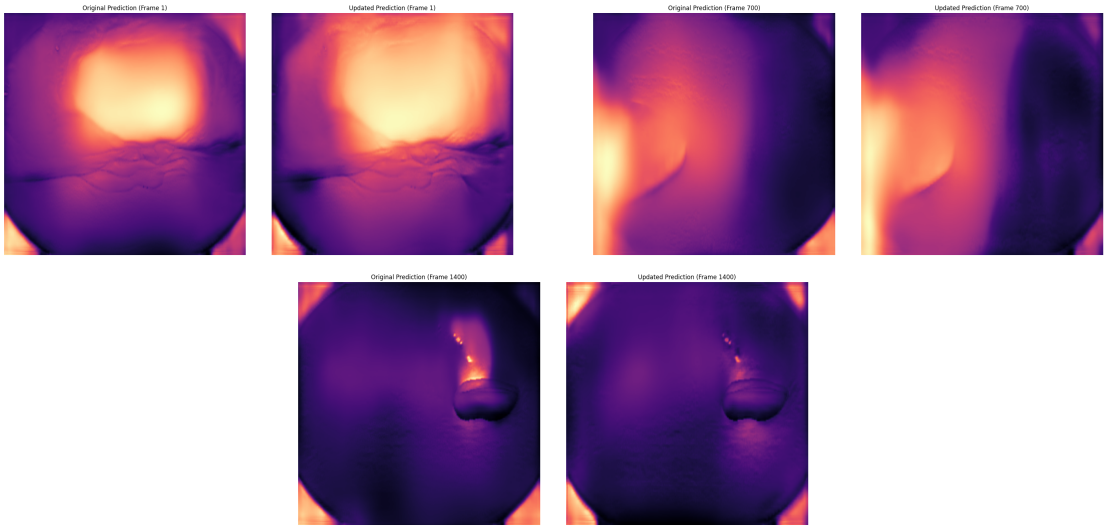


Figure 4.6: Visual comparison of predicted depth maps for three sample frames using the original and optimized inference pipelines.

In conclusion, the performance improvements introduced throughout the inference pipeline did not compromise the accuracy or visual quality of the depth maps, thereby validating the updated implementation for use in the final system.

4.2 3D Reconstruction Results - EndoSLAM Dataset

This section presents the 3D reconstruction results obtained using the optimized inference pipeline on the *EndoSLAM* dataset. The UnityCam subset was selected for evaluation, as it provides synthetic sequences of the stomach with per-frame ground truth depth maps and camera poses, which enable both qualitative and quantitative comparisons.

As a first step, depth maps were generated from the RGB input frames using the depth prediction module. These predictions were then compared against the corresponding ground truth depth maps, and subsequently projected into 3D point clouds. The projection was performed first into the camera coordinate frame using the intrinsic parameters, and then into the world coordinate frame using the associated camera poses. Finally, per-frame point clouds were aggregated over the sequence to reconstruct a full 3D model.

Unfortunately, the complete reconstruction could not be directly compared to a ground truth 3D model due to known inconsistencies and occlusions in the UnityCam GT mesh, which are discussed in detail later in this chapter.

For this dataset, the initial loading of RGB images, GT depth maps, and camera poses, as well as the predicted depth maps, was completed efficiently using the modular pipeline, taking approximately 0.3012 seconds.

4.2.1 Depth Maps Visualization and Evaluation

To ensure consistent visualization and fair evaluation between GT and predicted depth maps, a standardized pre-processing pipeline was applied to both sources. Although the specific operations differed slightly depending on the origin of the depth map, the overall objective was to align them in terms of scale, resolution, and valid field of view. While the complete procedure is detailed in Section 3.3.2, a concise summary is included below to aid interpretation:

- **Ground Truth Depth Maps:**
  - Inversion of depth values (since UnityCam stores depth inversely, with closer regions having higher values).
  - Conversion from centimeters to meters;
  - Application of a circular mask to restrict to valid endoscopic view.
- **Predicted Depth Maps:**
  - Resizing to match the resolution of the RGB input (to align with camera intrinsics);
  - Median scaling based on corresponding GT values (to address relative scale);
  - Application of the same circular mask to exclude irrelevant regions.
- **Visualization:** Both GT and predicted maps were normalized to a common scale before mask application for visual comparison.

This preparation process was efficient, and the `prepare_data` function output both the ground truth and predicted depth maps, each represented in camera and world coordinates, along with the corresponding visualization variables.

Figures 4.7 and 4.8 present qualitative examples for two representative frames of the UnityCam sequence. Each shows: (left) the RGB input, (center) the GT depth map, and (right) the predicted depth map.

In the depth maps, brighter areas represent regions that are farther from the camera, while darker tones indicate surfaces that are closer. As seen in the RGB images, this shading aligns well with the scene's actual geometry, confirming the validity of both prediction and visualization.

To quantitatively assess prediction accuracy, a set of standard evaluation metrics, including MAE, MSE, RMSE, PSNR, and SSIM, was computed by comparing the processed predicted depth

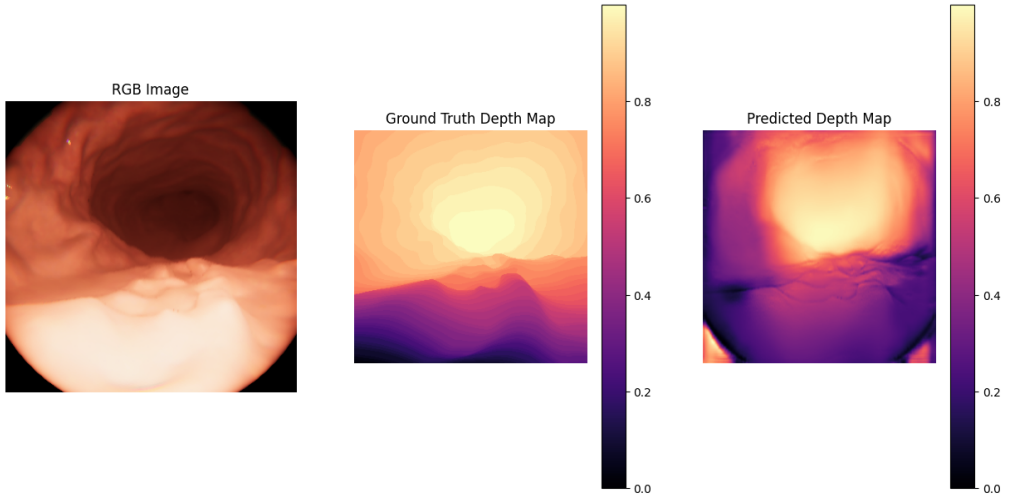


Figure 4.7: Frame 1 from the UnityCam sequence: RGB input, GT depth map, and predicted depth map.

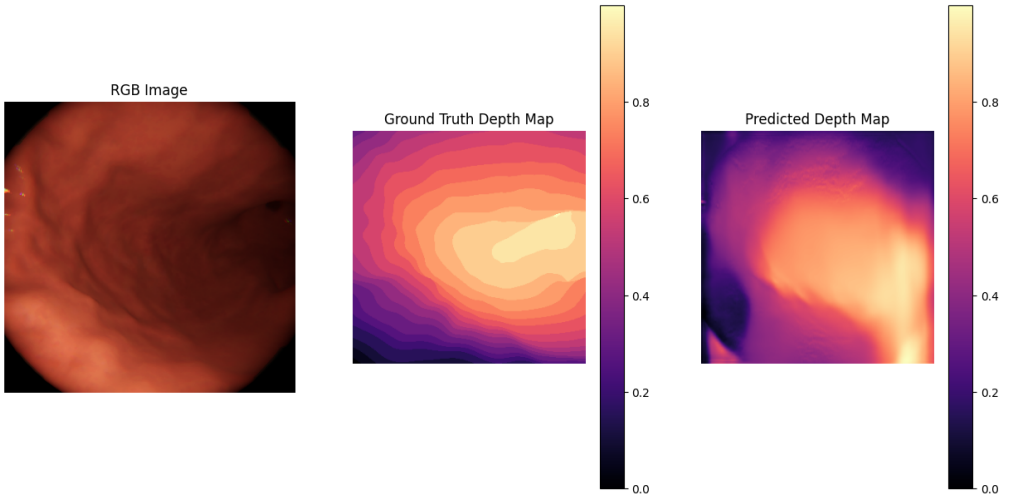


Figure 4.8: Frame 500 from the UnityCam sequence: RGB input, GT depth map, and predicted depth map.

maps to their corresponding ground truth (using only median scaling, without normalization). Table 4.9 reports the results for frames 1 and 500, representing early and mid-trajectory positions, along with the overall average values computed across all 1,548 frames, providing the global performance metrics.

The results highlight an expected improvement in depth estimation over time. Frame 1, at the start of the trajectory, shows higher errors, while Frame 500 exhibits notably better accuracy, reflecting the model's stabilization after initial frames.

Overall, these mean values indicate that the predictions are both accurate and structurally consistent with the ground truth. The relatively high SSIM score suggests that the model captures depth structure well, while the low RMSE and MAE reflect good pixel-wise correspondence.

Table 4.9: Comparison of predicted vs. ground truth depth maps for selected frames and overall average.

Metric	Frame 1	Frame 500	Mean (All Frames)
Mean Absolute Error (MAE)	0.0657	0.0220	0.0372
Mean Squared Error (MSE)	0.0066	0.0008	0.0031
Root Mean Squared Error (RMSE)	0.0811	0.0275	0.0498
Peak Signal-to-Noise Ratio (PSNR)	15.44 dB	16.79 dB	16.81 dB
Structural Similarity Index (SSIM)	0.823	0.832	0.833

In summary, the optimized inference pipeline produces high-quality depth predictions across the sequence, with low average error and noise (RMSE, MAE), high perceptual similarity (SSIM) and significant accuracy improvements between early and later frames. These results validate the suitability of the predicted depth maps for use in subsequent 3D reconstruction tasks.

4.2.2 Single-Frame Point Cloud Evaluation

Following depth map pre-processing, the corresponding point clouds were generated for individual frames. This section focuses on their visualization and evaluation, both in the camera and world coordinate frames, leaving the full-sequence 3D reconstruction analysis to subsequent sections.

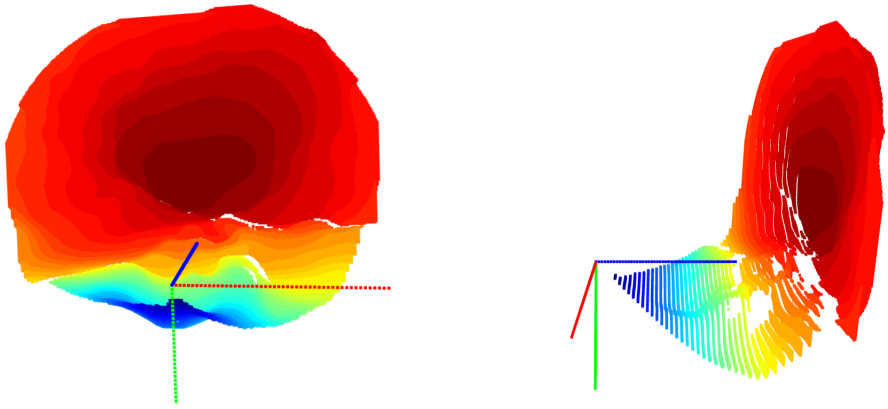
The processed depth maps already aligned in scale through pre-processing were converted into point clouds. The initial projection was into the camera coordinate frame using the known intrinsics (identical for both sources, as all sequences share the same synthetic camera). The resulting clouds were then transformed into the world coordinate frame using the corresponding camera poses.

Figures 4.9 and 4.10 show the resulting point clouds from the first frame in the camera frame. These view help confirm that the reconstructions are correctly aligned with depth along the z-axis—that is, the tube-like structure extends forward in the positive direction of the blue axis.

Analyzing these images, it becomes evident that the predicted point cloud exhibits the expected tube-like structure and extends well in depth along the z-axis. In contrast, the GT reconstruction appears noticeably flattened, displaying a clear lack of depth variation when projected into 3D, despite being synthetically generated. This may result from a dataset-related issue rather than a pre-processing error, as multiple correction strategies were tested without resolving the limitation.

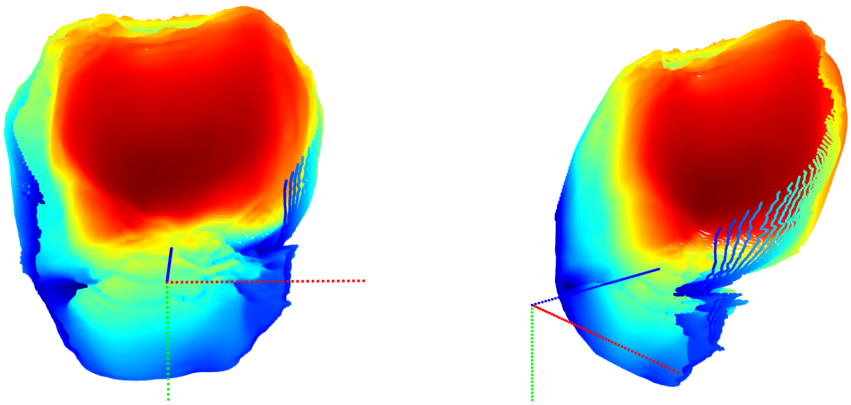
This compression in the GT geometry reduces its representativeness of the actual anatomical structure and weakens its utility for qualitative comparisons. While the ground truth still serves as a useful reference, these observations reinforce that it may not be a fully reliable geometric benchmark for evaluating 3D reconstruction quality.

To better isolate the tube structure typical of endoscopic scenes, a depth-based threshold was applied. Even in synthetic UnityCam sequences, the geometry mimics real endoscopy: a forward-facing tube with distant walls at the back of each frame. These distant points are not only less



(a) Front view (b) Side view

Figure 4.9: Ground truth point cloud in the camera coordinate frame. The structure follows the positive direction of the blue (z) axis.



(a) Front view (b) Side view

Figure 4.10: Predicted point cloud in the camera coordinate frame.

informative, but also potentially misleading for shape evaluation. To prevent this, only the closest 80% of points (based on depth percentiles) were retained. This removed the back wall and improved shape clarity.

The threshold was applied before axis reordering, while the point cloud was still aligned with the camera frame's native geometry (depth along z). To aid visualization, the following point clouds were downsampled by a factor of 0.02. Figure 4.12 shows the predicted cloud before (orange) and after thresholding (red). Figure 4.11 shows the same for the GT (blue and green).

Finally, Figure 4.13 overlays both thresholded point clouds (GT in green, prediction in red) for frame 1. The alignment appears qualitatively good, supporting the effectiveness of the depth predictions. As the pose data is shared between both sources, the world frame versions are not shown - the resulting reconstructions are nearly identical.

To complement the visual inspection, two geometric metrics were computed in the world frame:

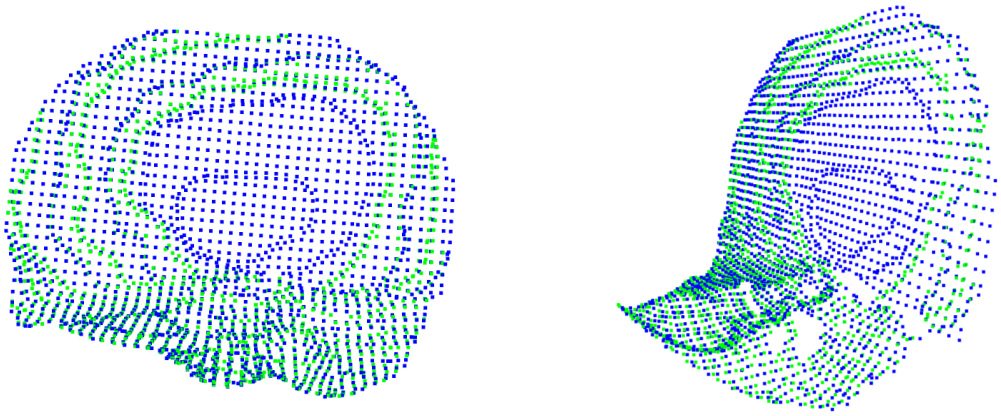


Figure 4.11: GT point cloud in the camera frame, showing the full (blue) and thresholded (green): front view (left), side view (right).

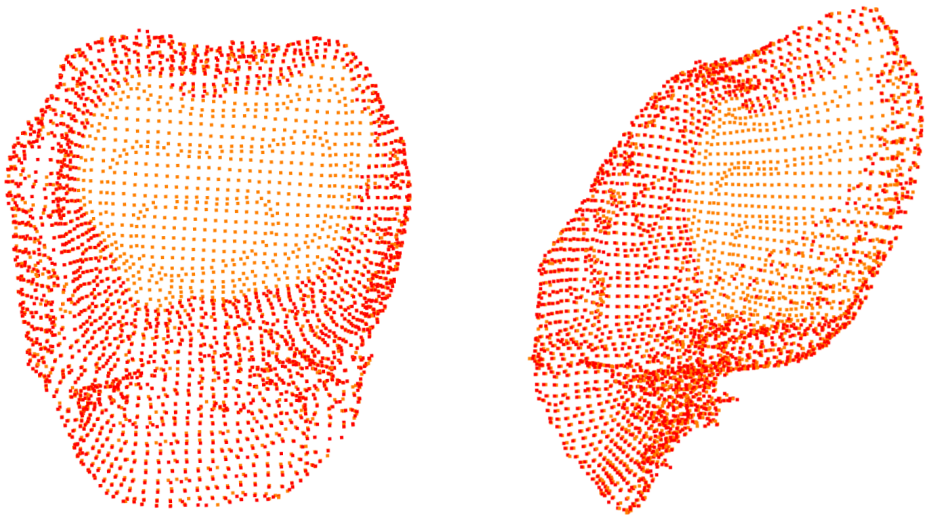


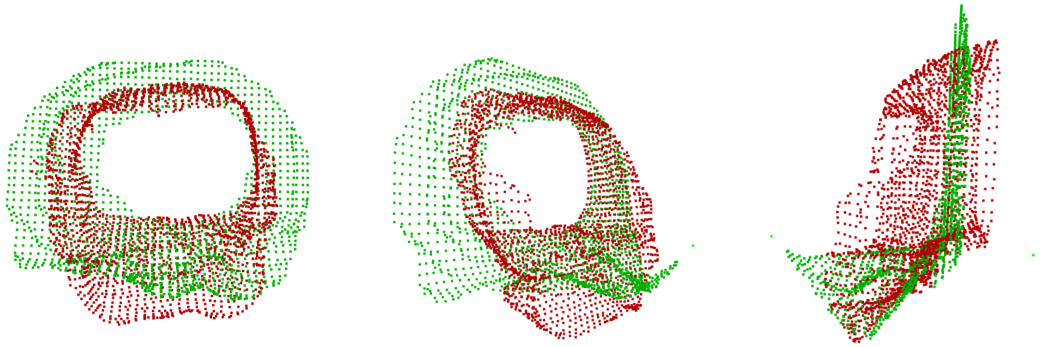
Figure 4.12: Predicted point cloud in the camera frame, showing the full cloud (orange) and the thresholded version (red): front view (left), side view (right).

Chamfer Distance and Hausdorff Distance. These provide frame-wise measurements of spatial similarity between the predicted and GT point clouds. Although originally calculated in meters, the results are reported here in centimeters for easier interpretation.

Table 4.10: Mean geometric distances across all frames comparing predicted and ground truth point clouds.

Metric	**Mean Value (cm)**
Chamfer Distance	6.38
Hausdorff Distance	13.75

Specifically, the Chamfer Distance reflects the average point-wise error between the predicted



(a) Front view (b) Side view (angled) (c) Side view (straight angle)

Figure 4.13: Overlay of predicted (red) and ground truth (green) point clouds in the camera frame: (a) front view, (b) side view (angled), and (c) side view (straight angle).

and ground truth surfaces, while the Hausdorff Distance captures the worst-case deviation. For both metrics, lower values indicate better geometric alignment.

Under this interpretation, and given the results computed over all 1,548 frames, a Chamfer Distance of 6.38 cm indicates that, on average, a point in one cloud lies approximately 6 cm from its nearest neighbor in the other. While this level of deviation is not negligible, it is reasonable in the context of monocular depth estimation, particularly when considering that the simulated stomach cavity spans approximately 10–20 cm.

As for the Hausdorff Distance, a value of 13.75 cm was obtained, corresponding to a maximum surface mismatch of nearly 14 cm. Although this represents a more significant local deviation, it is not entirely unexpected. The Hausdorff metric is highly sensitive to outliers and can be strongly influenced by known issues in the UnityCam ground truth point clouds.

To sum up, these findings were not unexpected. Given the limitations of the UnityCam ground truth, including the noticeable lack of depth variation when projected into 3D, a perfect one-to-one match was never anticipated. However, these limitations also make it difficult to extract truly meaningful conclusions from traditional accuracy metrics alone. From the visual analysis, it appears that if the GT data exhibited greater depth variation, it might align more closely with the predicted geometry.

Therefore, in this context, the goal is not to assess absolute precision, but rather to examine whether the predicted point clouds behave reasonably when compared to the available ground truth. Therefore, despite the GT's known issues, the qualitative overlays and moderate quantitative deviations suggest that the predictions are not arbitrarily wrong or distorted. While accuracy must be interpreted with caution, these comparisons still provide useful insight into the behavior of the depth prediction pipeline.

4.2.3 Full Point Cloud Reconstruction

As a final step, a complete 3D model was generated by combining the per-frame point clouds (in the world coordinate frame) across the entire sequence. This reconstruction was built by aggregating the spatially aligned point clouds over the trajectory.

Importantly, the median scaling step, previously applied during single-frame evaluation, was intentionally omitted here, as discussed in Section 3.3.2. There are two main reasons for this decision. First, the ground truth depth maps from the UnityCam dataset present known inconsistencies and artifacts, and transferring this potentially unreliable scale to the predicted data could degrade the results. In contrast, the predicted depth maps are more stable and consistent. Second, a practical 3D reconstruction pipeline should operate independently of ground truth, especially when aiming to reflect real-world deployment, where such references are typically unavailable.

To evaluate the effect of temporal sampling, the full reconstruction using all 1,548 frames was compared to a reduced version built using every 10th frame (resulting in approximately 155 frames). The visual and structural differences between the two models were minimal, and in practice, the reduced version was easier to interpret due to its lower density. This aligns with the nature of endoscopic imaging, where frame-to-frame differences are often small given the slow, continuous motion of the camera.

Considering the negligible visual benefit of using all frames and the significantly higher computational cost, the final 3D model was generated using the reduced version. This trade-off was applied consistently to both the predicted and ground truth reconstructions. Naturally, the depth thresholding step was applied throughout, as it remained essential to isolate the relevant tube-like anatomical structures.

Predicted 3D Model

To validate that a step size of 10 was sufficient, and even advantageous in terms of interpretability, the predicted 3D reconstruction was generated using two configurations: the full set of 1,548 frames and a reduced version constructed from every 10^{th} frame (155 in total).

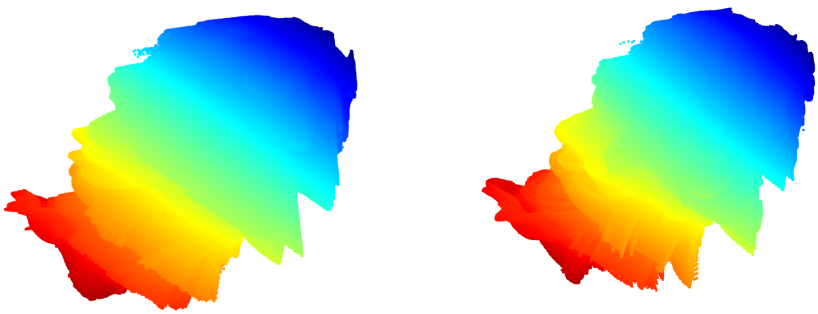


Figure 4.14: Comparison of predicted 3D point cloud models using all 1,548 frames (left) vs. every 10^{th} frame (right).

As shown in Figure 4.14, the reduced model preserved the overall structure, directionality, and spatial coherence of the full reconstruction. While it contains fewer frames (155 vs. 1,548) and significantly fewer points (approximately 10.3 million vs. 102.8 million), it remains visually faithful to the full version. Moreover, the reduced density makes the model easier to interpret, particularly in regions where frame redundancy offers little additional detail. For these reasons, the step-10 reconstruction was selected for subsequent analysis. It took only 5.95 seconds to generate, with an average processing rate of 26.03 FPS, offering a significant computational advantage without compromising visual fidelity.

To further illustrate the tubular structure, and, by extension, the anatomical plausibility, of the reconstructed model, Figure 4.15 presents a view from the initial portion of the trajectory, facing inward through the first visible opening. This perspective highlights the internal cavity of the predicted 3D point cloud, confirming that the reconstruction is indeed hollow, as expected for stomach-like anatomy.

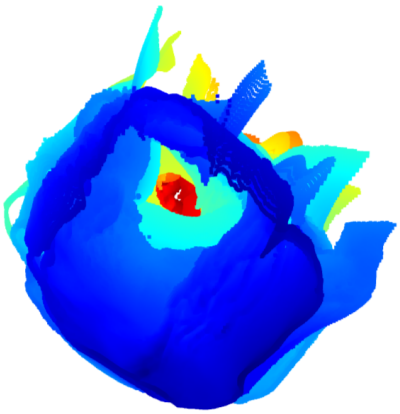


Figure 4.15: Internal view of the predicted 3D model showing its hollow, stomach-like structure.

To further improve visualization, optional downsampling was applied to the step-10 reconstruction, using a voxel size of 0.05. The result is shown in Figure 4.16.

Furthermore, upon inspecting the predicted 3D model, a few structural irregularities become apparent, such as spike-like artifacts protruding from the surface. These features are anatomically implausible and are likely caused by unreliable depth predictions in specific frames. One such example is frame 750, which exhibits clear prediction errors.

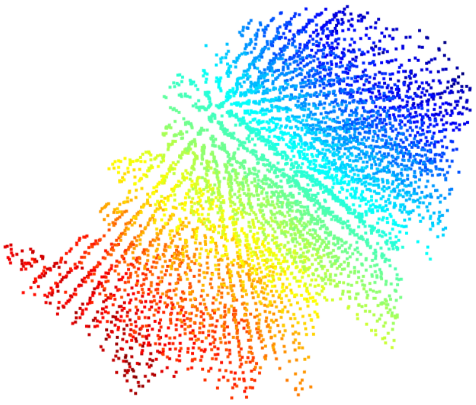


Figure 4.16: 3D reconstruction generated from predicted depth maps using a step size of 10, comprising 155 frames.

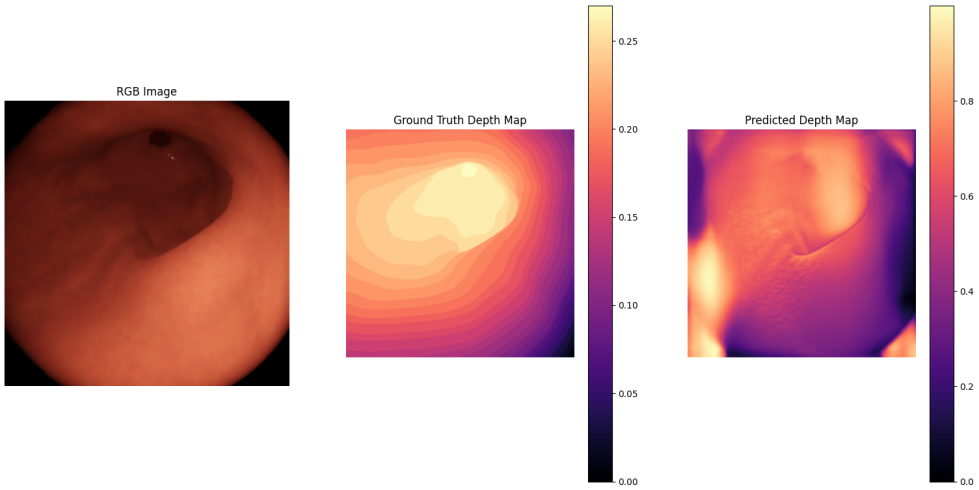


Figure 4.17: Frame 750: RGB input, ground truth depth map, and predicted depth map (left to right, respectively).

To illustrate this, Figure 4.17 presents the RGB input, ground truth depth map, and the corresponding predicted depth map for frame 750. It is visually evident that certain areas are incorrectly assigned excessively large depth values (highlighted in yellow), which should instead be more concentrated in the center or more evenly distributed along the left curve of the stomach. To highlight the local geometry contributing to the spikes observed in the full reconstruction, Figure 4.18 shows the thresholded point cloud generated from this frame alone, where irregular structures, especially near the left regions, can still be clearly observed despite the filtering.

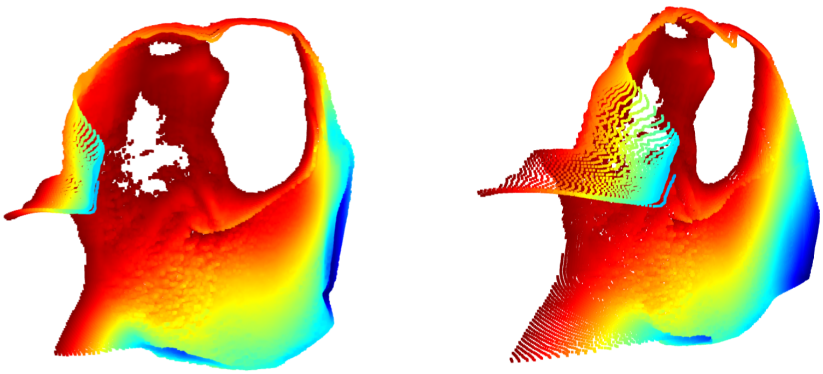


Figure 4.18: Point cloud generated from frame 750 (front and side views, respectively).

To contextualize the structure of the 3D model reconstructed from the *EndoSLAM* UnityCam sequence, Figure 4.19 presents an anatomical illustration of the stomach. The folds (rugae), concave curvature, and hollow cavity depicted in the illustration closely resemble the shape and surface patterns observed in the reconstructed point cloud. This visual similarity reinforces the idea that the predicted model captures plausible gastric-like geometry, which is an encouraging outcome given that the original dataset represents a simulated stomach environment. Although no ground truth 3D model exists to verify the reconstruction's accuracy, this qualitative alignment provides a small/meaningful reference and suggests that the pipeline is capable of generating anatomically realistic structures.

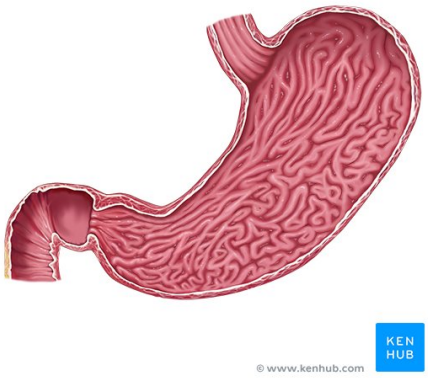


Figure 4.19: Anatomical illustration of the stomach [20].

Ground Truth 3D Model

For the ground truth data, a significant limitation was identified in the dataset. As discussed in Section 3.1.2.2, the GT depth maps are not considered reliable for meaningful evaluation of the predicted reconstructions due to inconsistencies in depth values along the trajectory. Additionally,

in Section 4.2.2, the single-frame point cloud analysis revealed another issue: the GT reconstructions exhibit a noticeable lack of depth variation, producing unnaturally flat structures even after correct pre-processing.

Nonetheless, to complete the pipeline and illustrate the output that would result from using the GT depth maps alone, a full 3D model was reconstructed. Following the same protocol used for the predicted model, the reconstruction was performed with a step size of 10 (i.e., 155 out of 1,548 frames), and a depth threshold of 80% was applied.

The resulting reconstruction is shown in Figure 4.20. It is visually evident that the GT model suffers from inconsistencies across frames, particularly in spatial coherence, which confirms the previously discussed limitations and reinforces the idea that these maps are not suited for reliable evaluation or benchmarking.

Figure 4.20: 3D reconstruction from GT depth maps using a step size of 10 (155 frames).

In conclusion, while the GT 3D model clearly shows depth-related issues, it does not offer much useful information beyond that. Due to its visual and structural inconsistencies, it cannot serve as a reliable reference for evaluating the predicted reconstruction.

Point Cloud 3D Models Registration

Despite having no real evaluative utility, a final step was performed in which both the predicted and ground truth point cloud models were registered in the same coordinate space to visualize their spatial relationship. The result is shown in Figure 4.21, with the corresponding geometric metrics reported below.

The Chamfer Distance was approximately 14.18 cm, while the Hausdorff Distance reached 34.27 cm.These large distances align with the visual differences observed in the overlay. Specifically, the GT model appears significantly larger and more spatially inconsistent than the predicted one, even though the predicted model already spans the full camera trajectory. This suggests that the GT geometry is not only less coherent but also misrepresents scale, reinforcing previous concerns raised in Section 3.1.2.2.

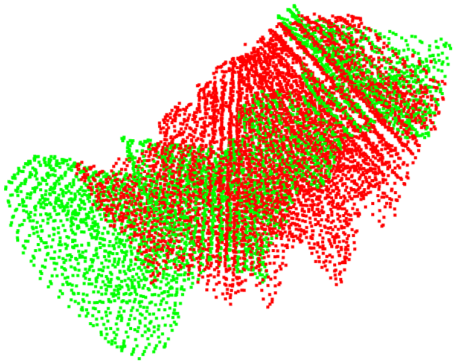


Figure 4.21: Overlay of predicted (red) and ground truth (green) 3D reconstructions on the EndoSLAM dataset.

In this context, the registration serves primarily as a qualitative illustration rather than a precise evaluation. The metrics highlight the scale mismatch and poor alignment, and further confirm that the GT model is not suitable as a reference for accurate geometric validation.

4.2.4 3D Mesh Reconstruction (Optional)

The surface reconstruction, which involved converting the 3D point cloud into a surface mesh, was considered an optional step, implemented primarily for visualization purposes or potential future applications. While it did not contribute significantly to the main analysis and was excluded from the core evaluation pipeline, it was still carried out for the predicted model to explore the feasibility of mesh generation.

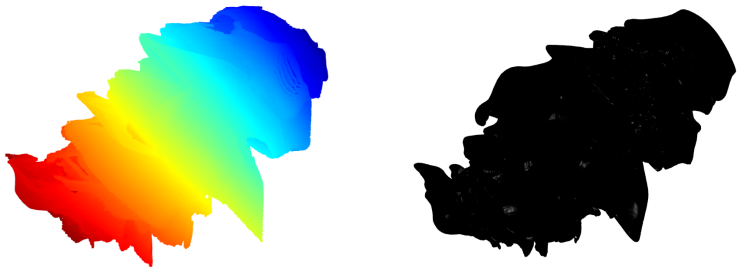
This process used the Poisson Surface Reconstruction technique and was applied only to the predicted model. The ground truth (GT) point cloud, due to its known inconsistencies, was not suitable for this operation and would not meaningfully benefit from mesh conversion.

Even when using the reduced predicted model (155 frames), the resulting point cloud was still too dense for direct mesh generation, causing memory issues and long processing times. To address this, an additional downsampling was performed using a step size of 100 (i.e., one out of every 100 frames), which preserved the global structure while significantly reducing the computational load.

This reconstruction produced two versions: a raw mesh and a post-processed mesh with minor artifacts removed. For the predicted surface reconstruction, Figure 4.22 shows the input point cloud and the resulting post-processed mesh.

Mesh generation took approximately 4.7 seconds, with an additional 4.3 seconds for post-processing. To assess geometric deviation, the Hausdorff Distance was computed between each mesh and the original point cloud: 0.16 for the raw mesh and 0.18 for the post-processed mesh.

Although the post-processed mesh introduced slightly more deviation, it offered improved surface continuity and may be preferable for visualization or export purposes. However, due to its limited analytical value in this pipeline, surface reconstruction remained an optional step and was not used for further evaluation or analysis.



(a) Point Cloud (Step 100) (b) Post-Processed Mesh

Figure 4.22: Surface reconstruction from predicted point cloud using Poisson reconstruction with step size 100. Left: input point cloud. Right: post-processed mesh.

4.3 3D Reconstruction Results - Uterus Phantom Dataset

For the *Uterus Phantom* dataset, a reconstruction pipeline similar to the one used for *EndoSLAM* was applied. This section presents the evaluation of the predicted depth maps and the resulting 3D reconstructions using the provided RGB input images and ground truth depth data.

After visually inspecting all available predicted outputs, none stood out as significantly more accurate or promising than the others. Therefore, one set of predictions was selected arbitrarily for analysis. The chosen sequence corresponds to the uterus phantom model oriented toward the robotic arm (approximately a 90º rotation to the right from the camera's perspective), which offers a clear view of the phantom structure.

For that, the dataset was first loaded and pre-processed following the modular structure described in Section 3.3.1. This includes the input RGB frames, the ground truth depth maps, the predicted depth maps, and the corresponding camera poses. The overall data loading and preparation step was efficient, taking approximately 0.7097 seconds.

Depth Maps Visualization

To enable visualization and comparison between the predicted and ground truth depth maps, all maps were pre-processed using the procedures previously described in Section 3.3.2. For the GT depth maps, values were first converted from millimeters to meters. Then, to reduce background interference and remove invalid readings, such as those representing space beneath the phantom table, a threshold of 0.6 m was applied.

As for the predicted depth maps, they were uncropped back to the original RGB resolution in order to match the GT dimensions and remain consistent with the camera intrinsics, since they had been cropped during model inference.

Finally, for visualization purposes only, both GT and predicted maps were normalized to a common value range to improve visual clarity and comparability.

A particular characteristic of this dataset is that some sequences were recorded along the same trajectory in both directions. This made it possible to compare frames captured from similar positions, but in opposite directions, one during forward motion and another during the return path. This setup allows an assessment of whether the predictions remain visually coherent when the same region is revisited.

To illustrate different points along the sequence and observe the model's prediction behavior, four representative frames were selected. Frames 1, 15, and 45 capture distinct views during the forward trajectory, while Frame 96 corresponds approximately to the same location as Frame 1 but captured on the return path, allowing for a direct comparison. The results are shown in the figures below, where each triplet includes the RGB input, the ground truth depth map, and the predicted depth map.

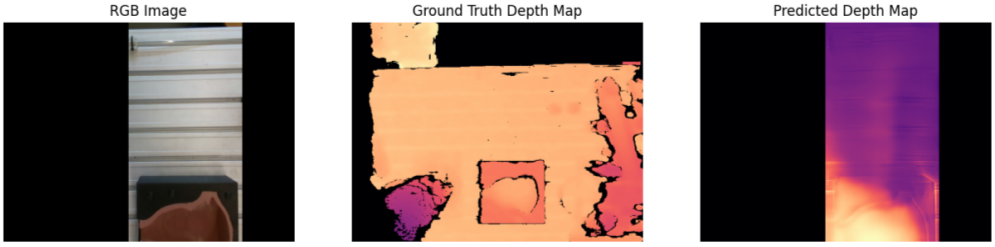


Figure 4.23: Depth estimation results for Frame 1. From left to right: RGB input image, GT depth map, and predicted depth map.

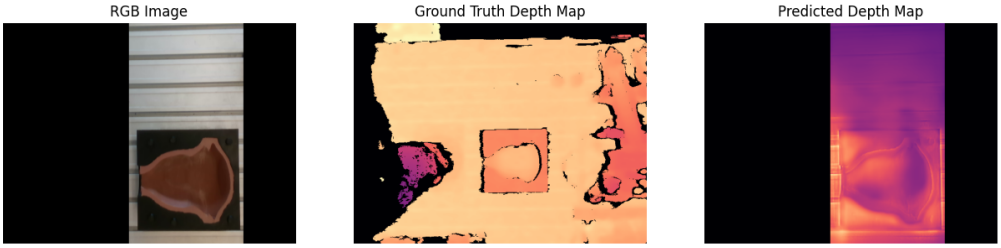


Figure 4.24: Depth estimation results for Frame 15. From left to right: RGB input image, GT depth map, and predicted depth map.

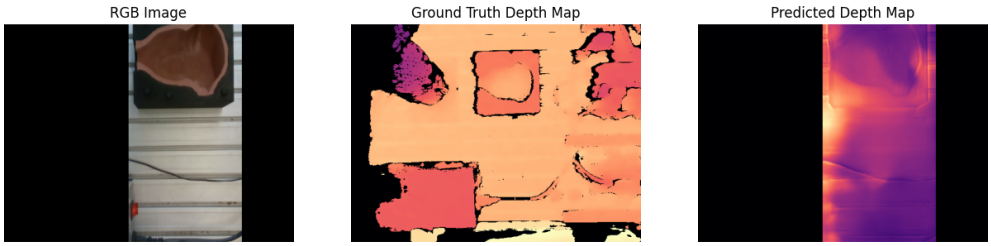


Figure 4.25: Depth estimation results for Frame 45. From left to right: RGB input image, GT depth map, and predicted depth map.

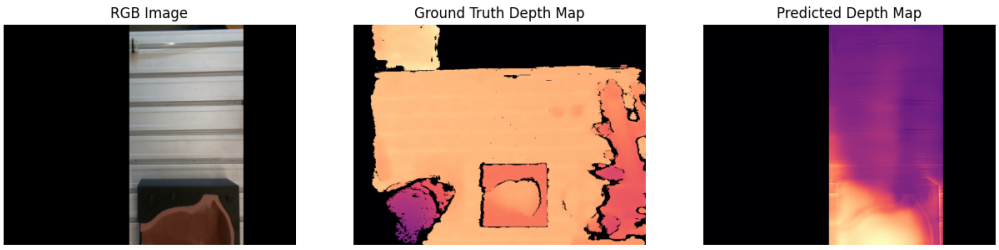


Figure 4.26: Depth estimation results for Frame 96 (reverse path). From left to right: RGB input image, GT depth map, and predicted depth map.

To evaluate the accuracy of the predicted depth maps, no metrics could be applied.

As previously explained in Section 3.1.2.2, the Intel RealSense device captures RGB and depth information using separate optical systems, which results in slightly different viewpoints and field-of-view distortions between the two streams. Notably, the RGB image appears zoomed-in compared to the depth map. This spatial mismatch means that the GT and predicted depth maps are not directly aligned, making per-pixel metric evaluation (e.g., MAE, MSE, SSIM) unreliable at this stage. However, after both sets of depth data are converted into 3D point clouds and projected into the same world coordinate frame, alignment and geometric comparison will be possible.

Nonetheless, visual inspection of the normalized depth maps already suggests that the predictions may not be reliable for the uterus phantom region. In multiple frames, a large concentration of high depth values (shown in bright yellow) appears across the model area. These elevated values indicate that the model is assigning excessively large distances to this region, which is inconsistent with the actual geometry of the phantom. Given that the uterus cavity is only around 7 cm deep, such depth exaggeration is clearly incorrect. Therefore, it likely reflects the model's poor generalization in a physical environment.

While no definitive conclusions can be drawn from pixel-wise comparisons, this qualitative evidence already points to systematic issues in the prediction of depth values for the uterus phantom portion of the scene.

Full Point Cloud Reconstruction

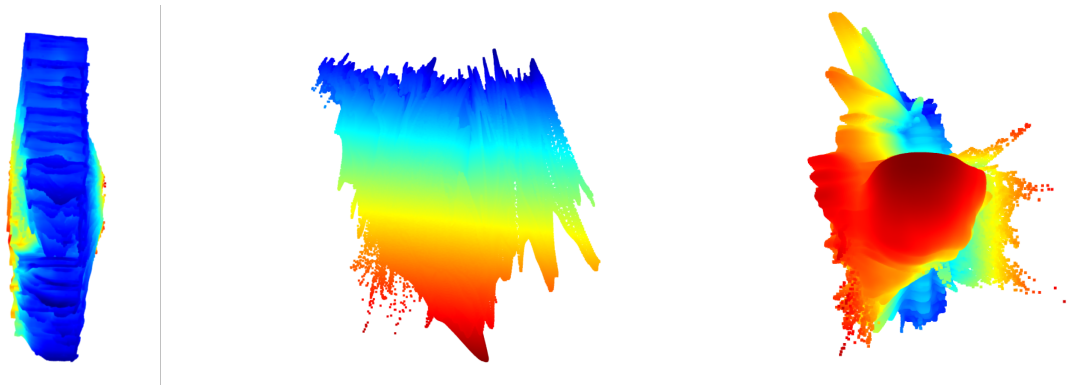
To visualize the 3D geometry from the depth maps, point clouds were generated, first in the camera coordinate frame and then transformed into the world frame. This was done using the intrinsic parameters of each camera (for RGB and GT depth) and the corresponding pose information from the trajectory. The reconstruction was performed for both the predicted and ground truth depth maps to enable a visual comparison of the resulting 3D models and highlight spatial consistency or deviations in the predicted geometry.

It is also important to note that single-frame visualizations (whether in the camera or world frame) were not emphasized in this dataset, as they did not offer any additional evaluation or meaningful insights. Instead, the focus here was placed on the final 3D reconstruction model.

Following the same logic as in the previous dataset, the reconstruction was performed using a step size of 10 to simplify processing and reduce redundancy. Many consecutive frames in the sequence were visually similar, and using all of them did not provide noticeable improvements in the final model.

The 3D model reconstruction was successfully completed. For the predicted depth maps, point cloud generation took approximately 4.81 seconds for 10 frames, with a processing speed of 2.08 FPS. For the ground truth depth maps, the process was slightly faster, achieving 3.83 seconds in total, corresponding to an average of 2.61 FPS. This small performance difference likely stems from the additional uncropping operations required for the predicted maps, which were originally resized during model inference.

Figures 4.27 and 4.28 present the resulting 3D point clouds from the predicted and GT depth maps, respectively. Multiple viewpoints are shown to facilitate a more complete understanding of the reconstructed geometry.



(a) Top view (b) Side view (c) Bottom view

Figure 4.27: Point cloud generated from the predicted depth map, viewed from multiple perspectives.

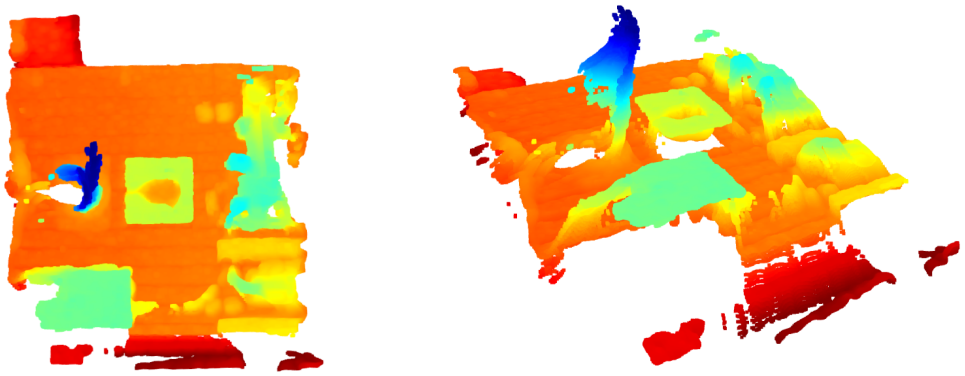


Figure 4.28: Point cloud generated from the ground truth depth map: top view (left), side view (right).

After generating the point clouds from both the predicted and ground truth depth maps, a registration step was applied to align them within the same 3D coordinate space. This enabled clearer visual comparison and geometric evaluation of how well the predicted model captured the true structure. To further simplify interpretation, the point clouds were downsampled using a voxel grid filter with a voxel size of `0.02`, which reduces point density while preserving the overall shape and structure. The result is shown in Figure 4.29, where the ground truth is displayed in red and the predicted model in green.

Overall, the alignment appears visually reasonable, with the predicted model well positioned within the larger GT cloud. However, due to the wider field of view of the ground truth sensor, the GT point cloud covers a much broader area, including regions not visible in the predicted data. This mismatch leads to inflated geometric error metrics: the Chamfer Distance was 24.66 cm and the Hausdorff Distance 48.21 cm—values that primarily reflect differences in spatial coverage rather than genuine structural misalignment.

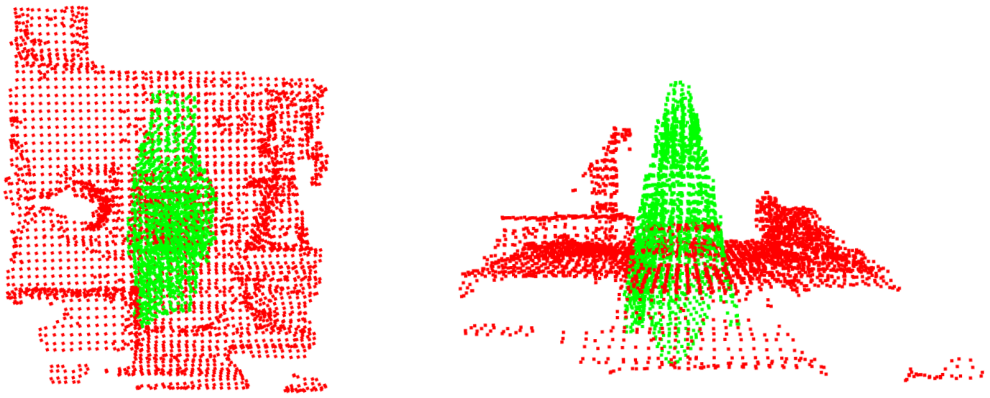


Figure 4.29: Registered point clouds showing alignment between ground truth (red) and predicted model (green): top view (left), front view (right).

In addition, the predicted depth values for the table surface are notably inaccurate, resulting in a surface that is incorrectly positioned, appearing above the GT reference (approximately 50 cm, which correctly corresponds to the actual distance between the camera and the physical table). This further highlights the model's difficulty in generalizing to this scene and undermines the reliability of the reconstructed geometry.

However, due to differences in camera setup, the ground truth depth maps have a noticeably wider field of view than the predicted ones. This causes the ground truth point cloud to include scene elements that are outside the predicted view. To ensure a fairer comparison, the ground truth point cloud was cropped to approximately match the spatial extent of the predicted model, as shown in Figure 4.30.

This adjustment improved both the visual alignment and the quantitative similarity, with the Chamfer Distance decreasing to 18.28 cm and the Hausdorff Distance to 30.93 cm, indicating a closer geometric match between the two models. However, even within this more targeted region, it remains evident that the prediction model did not generalize well to this dataset. The resulting 3D

reconstruction from the predictions exhibits visible artifacts, such as spikes and irregular surfaces, undermining its structural fidelity.

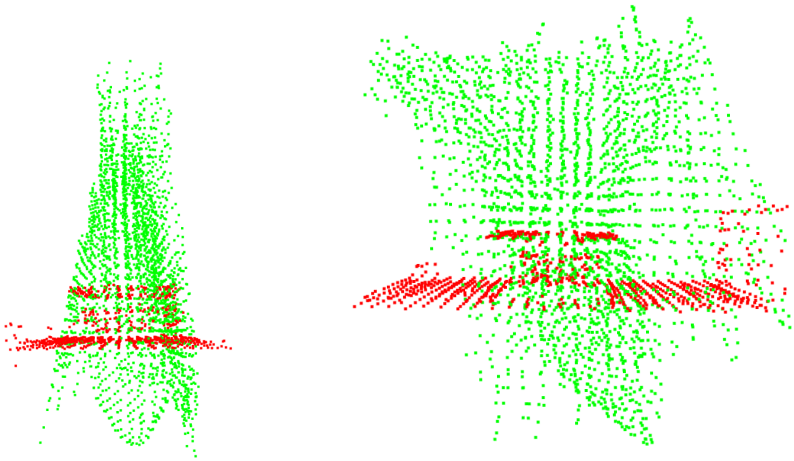
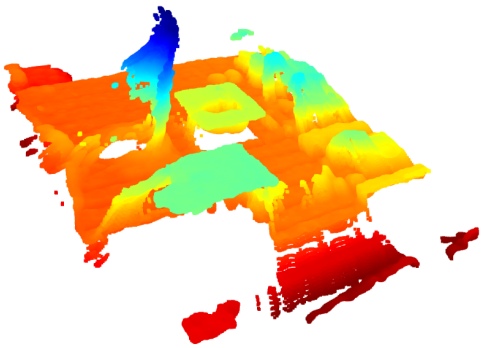


Figure 4.30: Registered point clouds after cropping the ground truth (green) to match the predicted view (red): front view (left), side view (right).

Therefore, despite the improved alignment in this cropped configuration, the overall outcome remains consistent with prior observations: the predictions are unreliable for this scene, and the resulting point cloud is too structurally inconsistent to support meaningful interpretation or evaluation.

A final observation worth highlighting is that the ground truth data produced an expected and structurally coherent result, especially considering the environment in which it was acquired, a fully controlled laboratory setting. Since the dataset was custom-built, this outcome confirms that the 3D reconstruction pipeline functions correctly when provided with consistent and reliable input.

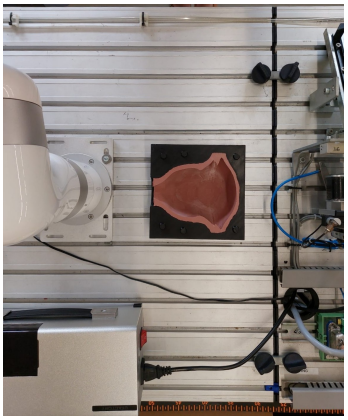


(a) 3D reconstruction from GT depth maps

(b) Laboratory setup used for data collection

Figure 4.31: Side view comparison between the GT-based 3D reconstruction and the real-world laboratory setup.

(a) 3D reconstruction from GT depth maps



(b) Laboratory setup used for data collection

Figure 4.32: Top view comparison between the GT-based 3D reconstruction and the real-world laboratory setup.

To reinforce this point, Figure 4.31 and Figure 4.32 show the reconstructed 3D model from the ground truth depth maps alongside photographs of the physical experimental setup, from side and top views, respectively. The visual alignment between the point cloud structure and the real-world scene supports the accuracy and validity of the reconstruction process.

4.4 Full Pipeline Performance

Considering the full pipeline as the combination of depth prediction and 3D reconstruction, the total execution speed reached approximately 8 FPS on the simulated stomach dataset from *EndoSLAM*, which closely resembles endoscopic and hysteroscopic image characteristics.

This value was computed by summing the individual execution times: 11.99 FPS for depth prediction and 26.03 FPS for 3D reconstruction (measured using step 10 applied over the frames). This step can be considered representative of a real-case scenario, as video data typically consists of a large number of consecutive frames. The final measurement incorporates all processing stages, from input pre-processing to final output generation.

Although small fluctuations in execution time may occur due to system load or input variability, this estimate provides a realistic benchmark. Therefore, we can conservatively state that the pipeline performs at approximately 8 FPS under realistic conditions.

Chapter 5

Conclusions and Future Work

This dissertation explored the feasibility and challenges of performing real-time 3D reconstruction of the uterine cavity from monocular input, by combining a neural network-based depth prediction module (adapted from the *EndoSLAM* framework) with a 3D reconstruction pipeline. The goal was not only to generate spatial models from RGB images but also to do so in a manner that approaches real-time performance.

A significant challenge encountered throughout this work was the lack of suitable datasets, particularly in the gynecological domain. The *EndoSLAM* dataset, specifically its simulated stomach subset, was initially adopted due to its anatomical and visual resemblance to the uterine cavity. However, inconsistencies in its ground truth depth maps limited the reliability of 3D reconstruction and hindered proper quantitative evaluation of depth prediction accuracy.

To address this limitation, a new dataset was created using a 3D-printed uterus phantom, with image and depth data captured using the Intel RealSense D435i. Although this setup offered greater control and predictable geometry, it came with important constraints, most notably, the camera's limited sensing range, which prevented close-up acquisition, and the absence of a ground truth pose tracking system. As a result, the data was collected from a wider field of view than ideal, and the depth predictions suffered from generalization issues. Consequently, while this dataset was not ideal for assessing the depth prediction module's robustness, it proved useful for validating the implementation and spatial consistency of the 3D reconstruction pipeline.

Due to the limitations of both datasets, a quantitative evaluation of depth prediction accuracy remained challenging. Nonetheless, the 3D reconstruction pipeline was successfully implemented, as verified by the generated point clouds aligning well with the known geometry of the phantom scene. This confirmed the spatial coherence of the reconstructed models under controlled conditions and demonstrated the feasibility of the 3D mapping approach.

Nevertheless, the speed optimization of the depth prediction module showed promising results. Optimization strategies, including OpenCV-based pre-processing, input downscaling, and batch processing, led to an approximate 50% reduction in inference time compared to the original *EndoSLAM* pipeline. This demonstrated that execution speed can be significantly improved without substantially compromising output quality, offering a potential path toward near real-time

applications in clinical environments.

The combined time for both depth prediction and 3D reconstruction reached approximately 8 FPS on the simulated stomach dataset, which is below typical real-time requirements, especially considering that hysteroscopic video streams generally operate at 25–30 Hz [25]. Under these conditions, the system does not yet achieve real-time 3D feedback. Nonetheless, its current performance represents a significant step forward, and further optimizations may enable real-time operation in the future. However, a key limitation remains: the pipeline still relies on ground truth camera poses, which prevents full online applicability. To achieve a truly real-time system, it will be necessary to incorporate a visual odometry or SLAM-based module for pose estimation.

Based on the findings and limitations discussed above, several directions for future research and improvement emerge:

- Eliminate the reliance on ground truth poses by integrating a visual odometry module, such as EndoSLAM's pose estimation network. This would enable full real-time 3D reconstruction during surgery.
- Enhance depth prediction with temporal consistency, leveraging recurrent or multi-frame approaches to reduce flickering and improve stability across video frames.
- Fine-tune existing models (e.g., DispResNet or MiDaS) on domain-specific data or explore lightweight alternatives like FastDepth and depth prediction GANs to boost both accuracy and speed.
- If physical phantoms are used again, improvements in hardware acquisition are necessary - ideally using a resin printer for finer detail and a depth camera capable of capturing accurate RGB-D data at close distances, with an integrated tracking sensor.
- However, a more effective solution may lie in transitioning toward high-quality synthetic datasets, such as Unity-based simulations, or acquiring access to annotated hysteroscopic video with clinical-grade annotations.

Although the proposed pipeline is currently executed offline due to the lack of real-time pose estimation, its modular and optimized structure makes it well-suited for future integration into real-time systems. With the addition of SLAM-based localization and appropriate hardware acceleration, the approach could evolve into a practical tool for intraoperative 3D mapping, contributing to surgical guidance and supporting clinical decision-making.

Appendix A

Uterus Phantom Dataset Creation

This appendix describes the creation and organization of the uterus phantom dataset used in this work. A custom script was implemented to extract RGB and depth data from RealSense `.bag` files, synchronize the modalities, and compute camera poses based on robotic arm motion.

A.1 Data Organization Script

The following script was developed in Python using the `pyrealsense2` library to extract and organize data from the RealSense `.bag` recordings. It saves RGB images, depth maps, and computes camera poses based on the robotic arm's motion parameters.

```
import pyrealsense2 as rs
import numpy as np
import cv2
import pandas as pd
import os
import math

bag_file = r"C:\Users\ASUS\Desktop\TESE\new_dataset_feup\to_analize\front_view.bag"
output_dir = r"C:\Users\ASUS\Desktop\TESE\new_dataset_feup\analized\
    front_view_data_updated"

# Define the first frame -  "left_view.bag"
#first_depth_frame = 10988 # for the depth
#first_rgb_frame = 10991 # for the RGB

# Define the first frame -  "front_view.bag"
first_depth_frame = 4974 # for the depth
first_rgb_frame = 4977 # for the RGB

# To crop the RGB images
crop_left_fraction = 0.43
crop_right_fraction = 0.18
```

```
22
23 ##### CAMERA AND ROBOT INFORMATION #####
24
25 # Robot motion parameters
26 robot_x = 200
27 robot_z = 400
28 roll, pitch, yaw = 180, 0, 86 # orientation data in Euler angles - RPY order
29
30 # Motion configuration
31 frames_forward = 48 # 45 # frames (seconds*FPS) = 3*14.99 = 44.97 ~ 45
32 pause_frames = 79 # 75 # 5 seconds * 14.99 FPS = 74.95 ~ 75 frames
33 frames_backward = 48 # 45
34
35 # Total expected frames to track
36 frame_ids_to_capture = set()
37 for i in range(frames_forward):
38     frame_ids_to_capture.add(first_depth_frame + i)  # forward
39 for i in range(frames_backward):
40     frame_ids_to_capture.add(first_depth_frame + frames_forward + pause_frames + i)
          # backward
41
42 # Create output folders
43 rgb_dir = os.path.join(output_dir, "rgb")
44 depth_dir = os.path.join(output_dir, "depth")
45 os.makedirs(rgb_dir, exist_ok=True)
46 os.makedirs(depth_dir, exist_ok=True)
47
48 ##### PIPELINE SETUP #####
49
50 pipeline = rs.pipeline()
51 config = rs.config()
52 rs.config.enable_device_from_file(config, bag_file)
53 config.enable_stream(rs.stream.depth)
54 config.enable_stream(rs.stream.color)
55 pipeline.start(config)
56
57 # To check the depth scale
58 #depth_sensor = pipeline.get_active_profile().get_device().first_depth_sensor()
59 #depth_scale = depth_sensor.get_depth_scale()
60 #print("Depth Scale:", depth_scale)
61
62 frames_to_save = {
63     "tX": [],
64     "tY": [],
65     "tZ": [],
66     "eX": [],
67     "eY": [],
68     "eZ": [],
69     "time(s)": []
```

```
}

frames_to_save_more_info = {
    "depth_frame_id": [],
    "rgb_frame_id": [],
    "depth_timestamp(s)": [],
    "rgb_timestamp(s)": [],
    "tX": [],
    "tY": [],
    "tZ": [],
    "eX": [],
    "eY": [],
    "eZ": [],
    "direction": []
}

first_ts = None
rgb_counter = 0
depth_counter = 0

try:
    while True:
        frames = pipeline.wait_for_frames()
        color_frame = frames.get_color_frame()
        depth_frame = frames.get_depth_frame()

        if not depth_frame or not color_frame:
            continue

        depth_frame_id = depth_frame.get_frame_number()
        rgb_frame_id = color_frame.get_frame_number()

        if depth_frame_id in frame_ids_to_capture:
            # Determine direction
            if depth_frame_id < first_depth_frame + frames_forward:
                direction = "forward"
                index = depth_frame_id - first_depth_frame
                y = 150 - index * (300 / (frames_forward - 1))
            else:
                direction = "backward"
                index = depth_frame_id - (first_depth_frame + frames_forward +
                    pause_frames)
                y = -150 + index * (300 / (frames_backward - 1))

            # Get timestamp
            ts = depth_frame.get_timestamp() / 1000.0  # convert ms to seconds
            ts_depth = depth_frame.get_timestamp() / 1000.0  # convert ms to
                seconds
            ts_rgb = color_frame.get_timestamp() / 1000.0  # convert ms to seconds
```

```
117
118             # Convert images
119             depth_image = np.asanyarray(depth_frame.get_data())
120
121             color_image = np.asanyarray(color_frame.get_data())
122             #print("Depth image shape:", depth_image.shape)
123             #print("Color image shape:", color_image.shape)
124
125             # Convert RGB to BGR for OpenCV saving
126             color_image_rgb = cv2.cvtColor(color_image, cv2.COLOR_RGB2BGR)
127
128             # Asymmetric crop for RGB
129             w_rgb = color_image.shape[1]
130             crop_left_rgb = int(w_rgb * crop_left_fraction)
131             crop_right_rgb = w_rgb - int(w_rgb * crop_right_fraction)
132             cropped_rgb = color_image_rgb[:, crop_left_rgb:crop_right_rgb, :]
133
134             # Save images with custom filenames
135             rgb_filename = f"image_{rgb_counter:04d}.png"
136             depth_filename = f"aov_image_{depth_counter:04d}.png"
137             cv2.imwrite(os.path.join(rgb_dir, rgb_filename), masked_rgb)
138             cv2.imwrite(os.path.join(depth_dir, depth_filename), depth_image)
139             rgb_counter += 1
140             depth_counter += 1
141
142             # Save pose with both frame IDs and timestamps
143             if first_ts is None:
144                 first_ts = ts
145             frames_to_save["tX"].append(robot_x)
146             frames_to_save["tY"].append(round(y, 2))
147             frames_to_save["tZ"].append(robot_z)
148             frames_to_save["eX"].append(roll)
149             frames_to_save["eY"].append(pitch)
150             frames_to_save["eZ"].append(yaw)
151             frames_to_save["time(s)"].append(ts - first_ts)
152
153             # Save pose with both frame IDs and timestamps
154             frames_to_save_more_info["depth_frame_id"].append(depth_frame_id)
155             frames_to_save_more_info["rgb_frame_id"].append(rgb_frame_id)
156             frames_to_save_more_info["depth_timestamp(s)"].append(ts_depth)
157             frames_to_save_more_info["rgb_timestamp(s)"].append(ts_rgb)
158             frames_to_save_more_info["tX"].append(robot_x)
159             frames_to_save_more_info["tY"].append(round(y, 2))
160             frames_to_save_more_info["tZ"].append(robot_z)
161             frames_to_save_more_info["eX"].append(roll)
162             frames_to_save_more_info["eY"].append(pitch)
163             frames_to_save_more_info["eZ"].append(yaw)
164             frames_to_save_more_info["direction"].append(direction)
165
```

```
166         # Early exit once all target frames are captured
167         if len(frames_to_save_more_info["depth_frame_id"]) >= (frames_forward +
                frames_backward):
168             break
169 
170 except Exception as e:
171     print("Finished or error:", e)
172 
173 pipeline.stop()
174 
175 ##### SAVE CSV FILE #####
176 
177 df = pd.DataFrame(frames_to_save)
178 csv_path_test = os.path.join(output_dir, "poses.csv")
179 df.to_csv(csv_path_test, index=False)
180 
181 df_more_info = pd.DataFrame(frames_to_save_more_info)
182 csv_path = os.path.join(output_dir, "poses_more_info.csv")
183 df_more_info.to_csv(csv_path, index=False)
184 
185 print(f"Done! Saved {len(df)} frames and corresponding poses.")
```

Listing A.1: Data extraction and organization script for the uterus phantom dataset.

Appendix B

Depth Prediction Module

This appendix details the implementation of the depth estimation module used in this work, including the model definition, inference script, and supporting functions.

B.1 Model Initialization Code

The code below defines the final version of the depth prediction model used in this work. It combines a ResNet-based encoder with a decoder into a unified architecture, `DispResNet`, and includes a helper function to load pretrained weights.

```
import torch
import torch.nn as nn
from .encoder import ResnetEncoder
from .decoder import DepthDecoder

class DispResNet(nn.Module):
    def __init__(self, num_layers=18, pretrained=True):
        super().__init__()
        self.encoder = ResnetEncoder(
            num_layers=num_layers,
            pretrained=pretrained,
            num_input_images=1
        )
        self.decoder = DepthDecoder(self.encoder.num_ch_enc)

    def forward(self, x):
        features = self.encoder(x)
        outputs = self.decoder(features)
        return outputs[0] if not self.training else outputs

def load_pretrained(weight_path="pretrained/pretrained_dispnet.pth", map_location=
    None, device=None, num_layers=18):
    if device is None:
```

```
        device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    if map_location is None:
        map_location = device
    model = DispResNet(num_layers=num_layers, pretrained=False)
    state_dict = torch.load(weight_path, map_location=map_location)
    model_state = state_dict.get('state_dict', state_dict)
    model.load_state_dict(model_state)
    return model.eval().to(device)
```

Listing B.1: Simplified depth prediction model definition combining encoder and decoder.

B.2 Optimized Inference Script

The Python script below implements the final version of the depth prediction pipeline used in this work. Key improvements include OpenCV-based pre-processing, input downscaling, batch inference, and optional mixed precision support.

Note: For clarity and readability, auxiliary timing code and commented-out testing logic have been omitted from the version shown here. The script presented focuses on the core functionality only.
Note: Standard imports (e.g., `cv2`, `torch`, `numpy`) omitted for brevity.

Script name: `inference.py`, located in the `updated_depth_predictions/` directory.

```
from tqdm import tqdm
from models import load_pretrained
from utils.transforms import preprocess_image

def process_folder(
    input_folder: str,
    output_folder: str,
    weight_path=r"C:\Users\ASUS\Desktop\TESE\CODE\Codigo_tese\src\
        updated_depth_predictions\pretrained\pretrained_dispnet.pth",
    batch_size: int = 8
):

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = load_pretrained(weight_path, num_layers=18).to(device)
    model.eval()

    os.makedirs(output_folder, exist_ok=True)

    image_files = [f for f in os.listdir(input_folder) if f.lower().endswith(('.jpg
        ', '.png'))]
    all_depth_maps = []
    all_filenames = []
```

```
21
22      # Batch processing
23      for batch_idx in tqdm(range(0, len(image_files), batch_size)):
24          img_files = image_files[batch_idx:batch_idx+batch_size]
25          batch_tensors = []
26
27          #preprocess_start = time.time()
28          for img_file in img_files:
29              input_path = os.path.join(input_folder, img_file)
30              # Downscaling (75%)
31              img_tensor = preprocess_image(input_path, target_size=(624, 192))
32              batch_tensors.append(img_tensor)
33          batch_tensor = torch.stack(batch_tensors, dim=0).to(device)
34
35          # Mixed Precision
36          with torch.no_grad():
37              if device.type == "cuda":
38                  with torch.cuda.amp.autocast():
39                      disparity_maps = model(batch_tensor).cpu().numpy()
40              else:
41                  disparity_maps = model(batch_tensor).cpu().numpy()
42
43          depth_maps = 1.0 / (disparity_maps + 1e-6)
44
45          all_depth_maps.extend([depth_maps[i].squeeze() for i in range(len(img_files
                ))])
46          all_filenames.extend(img_files)
47
48      np.save(os.path.join(output_folder, "updated_predictions.npy"), np.array(
            all_depth_maps))
49
50  if __name__ == "__main__":
51
52      process_folder(
53          input_folder=r"C:\Users\ASUS\Desktop\TESE\CODE\Datasets\EndoSlam\UnityCam\
                Stomach\Frames",
54          output_folder=r"C:\Users\ASUS\Desktop\TESE\CODE\Results\Depth_Predictions",
55          batch_size=8
56      )
```

Listing B.2: Optimized inference script

```
1   def preprocess_image(image_path, downscale_factor=1.0, target_size=None):
2       img = cv2.imread(image_path)
3       img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
4
5       # Downscale
6       if target_size is not None:
```

```
        img = cv2.resize(img, target_size)  # (width, height)
    elif downscale_factor != 1.0:
        new_w = int(img.shape[1] * downscale_factor)
        new_h = int(img.shape[0] * downscale_factor)
        img = cv2.resize(img, (new_w, new_h))
    else:
        img = cv2.resize(img, (832, 256))  # default

    img = img.astype(np.float32)
    img = np.transpose(img, (2, 0, 1))
    img_tensor = torch.from_numpy(img)
    img_tensor = (img_tensor / 255.0 - 0.45) / 0.225  # Normalization using
        ImageNet stats
    return img_tensor
```

Listing B.3: Image preprocessing function used in the inference pipeline.

Appendix C

3D Reconstruction Pipeline

This appendix presents the core components of the 3D reconstruction pipeline developed for this thesis. It includes dataset loaders, pose parsing logic, depth map pre-processing functions, and routines for point cloud generation and transformation.

C.1 Data Loading and Reading

This section presents the data-loading routines used to access and structure the information required for 3D reconstruction. Two datasets were handled: the EndoSLAM UnityCam (simulated stomach) dataset and the uterus phantom dataset. In both cases, the data consisted of RGB frame sequences, depth maps, and corresponding camera pose data, all of which needed to be synchronized and structured for later processing.

To streamline access and pre-processing, each dataset was wrapped in a custom Python class using an object-oriented approach. These classes encapsulate the logic for reading individual components (images, depth maps, poses) and assembling them into a consistent format suitable for the reconstruction pipeline.

Note: The following classes assume the following libraries are imported:

```
import os
import pandas as pd
import cv2
import numpy as np
from PIL import Image
```

Listing C.1: Common imports used across all data handling classes.

C.1.1 EndoSLAM Dataset

The EndoSLAM UnityCam stomach dataset uses RGB images named in the format `image_0000.png` and depth maps named `aov_image_0000.png`. Pose data is stored in a CSV file containing

translation and rotation values, typically represented as quaternions.

```
# Class to read and process frame images (rgb)
class Frame_handler: # class definition
    def __init__(self, path_frames): # initialization method - path to the directory where RGB image frames are stored
        self.path_frames = path_frames

    def __len__(self): # length method - returns the number of frame images
        return len([f for f in os.listdir(self.path_frames) if f.startswith("image_") and f.endswith(".png")])
        #return len(os.listdir(self.path_frames)) # counting the files in that directory

    def get_frame(self, index): # get frame method - reads a frame image by index
        image_name = f"image_{index:04d}.png"  # format: image_0000.jpg -- for the EndoSLAM dataset
        path_image = os.path.join(self.path_frames, image_name) # joins each image with the path
        return Image.open(path_image).convert('RGB')  # Open image with PIL and convert to RGB
        #return cv2.imread(path_image) # read the image with imread

# Class to read and process depth maps
class Depth_map_handler:
    def __init__(self, path_depth_maps):
        self.path_depth_maps = path_depth_maps

    def __len__(self):
        return len([f for f in os.listdir(self.path_depth_maps) if f.startswith("aov_image_") and f.endswith(".png")])

    def get_depth_map(self, index):
        depth_map_name = f"aov_image_{index:04d}.png"  # format: aov_image_0000.png
        path_depth_map = os.path.join(self.path_depth_maps, depth_map_name)
        gt_gray = cv2.imread(path_depth_map, cv2.IMREAD_UNCHANGED)
        if gt_gray.ndim == 3 and gt_gray.shape[2] == 4:
            gt_gray = gt_gray[:, :, 0]  # Use only the first channel
        return gt_gray

# Class to load and access pose data (in meters)
class Pose_handler:
    def __init__(self, path_poses):
        self.poses = pd.read_csv(path_poses) # reads the CSV file into a pandas DataFrame (pandas automatically treats the first row as the header)
        self.poses.columns = self.poses.columns.str.strip() # strips any whitespace from column names

```

```
38      def __len__(self):
39          return len(self.poses)
40
41      def get_pose(self, index): # fetches pose data for a given index
42
43          pose_data = self.poses.iloc[index] # retrieves the row at the specified
                index
44          # Check if quaternion columns exist
45          if all(col in pose_data for col in ['rX', 'rY', 'rZ', 'rW']):
46              return { # returns a dictionary with the pose's translation and
                    rotation components
47                  "trans_x": pose_data['tX'],
48                  "trans_y": pose_data['tY'],
49                  "trans_z": pose_data['tZ'],
50                  "quot_x": pose_data['rX'],
51                  "quot_y": pose_data['rY'],
52                  "quot_z": pose_data['rZ'],
53                  "quot_w": pose_data['rW'],
54              }
55          # Otherwise, use Euler angles
56          elif all(col in pose_data for col in ['eX', 'eY', 'eZ']):
57              return {
58                  "trans_x": pose_data['tX'],
59                  "trans_y": pose_data['tY'],
60                  "trans_z": pose_data['tZ'],
61                  "eul_x": pose_data['eX'],
62                  "eul_y": pose_data['eY'],
63                  "eul_z": pose_data['eZ'],
64              }
65          else:
66              raise KeyError("Pose CSV does not contain expected rotation columns (rX
                    /rY/rZ/rW or eX/eY/eZ)")
67
68  # Class for the dataset, integrating frames, depth maps, and poses
69  class Dataset_unity_cam:
70      def __init__(self, path_frames, path_depth_maps, path_poses):
71          self.frames = Frame_handler(path_frames)
72          self.depth_maps = Depth_map_handler(path_depth_maps)
73          self.poses = Pose_handler(path_poses)
74          self.length = min(len(self.frames), len(self.depth_maps), len(self.poses))
                # use the minimum available # set length to 1543 (number of valid poses
                )
75
76      def __len__(self): # returns the number of samples in the dataset
77          return self.length
78
79      def __getitem__(self, i): # returns a dictionary containing image, depth map,
            and pose for the given index
80
```

```
        return {
            "Image": self.frames.get_frame(i),
            "Depth_map": self.depth_maps.get_depth_map(i),
            "Pose": self.poses.get_pose(i),
        }

    # To easily access all the GT depth maps
    def get_all_depth_maps(self):
        return [self.depth_maps.get_depth_map(i) for i in range(len(self))]
```

Listing C.2: Dataset handler classes for the EndoSLAM UnityCam subset.

C.1.2 Uterus Phantom Dataset

The uterus phantom created dataset follows a similar structure to the EndoSLAM UnityCam subset but introduces a few key differences. Pose data is provided exclusively in Euler angles, and depth maps are stored as 16-bit PNG files. Additionally, translation values are originally in millimeters and are converted to meters, while the rotation angles are converted from degrees to radians. Ground truth depth map values are also in millimeters and undergo conversion to meters during pre-processing.

```
# Class to read and process frame images (rgb)
class Frame_handler:
    def __init__(self, path_frames):
        self.path_frames = path_frames

    def __len__(self):
        return len([f for f in os.listdir(self.path_frames) if f.startswith("image_
            ") and f.endswith(".png")])

    def get_frame(self, index):
        image_name = f"image_{index:04d}.png"
        path_image = os.path.join(self.path_frames, image_name)
        return Image.open(path_image).convert('RGB')

# Class to read and process depth maps (provided in mm but converted later)
class Depth_map_handler:
    def __init__(self, path_depth_maps):
        self.path_depth_maps = path_depth_maps

    def __len__(self):
        return len([f for f in os.listdir(self.path_depth_maps) if f.startswith("
            aov_image_") and f.endswith(".png")])

    def get_depth_map(self, index):
        depth_map_name = f"aov_image_{index:04d}.png"
```

```
24          path_depth_map = os.path.join(self.path_depth_maps, depth_map_name)
25          gt_gray = cv2.imread(path_depth_map, cv2.IMREAD_UNCHANGED) # preserves 16-
                bit depth
26          return gt_gray
27
28  # Class to load and access pose data (convert from mm to meters and from degrees to
        radians)
29  class Pose_handler:
30      def __init__(self, path_poses):
31          self.poses = pd.read_csv(path_poses) # reads the CSV file into a pandas
                DataFrame (pandas automatically treats the first row as the header)
32          self.poses.columns = self.poses.columns.str.strip() # strips any whitespace
                 from column names
33
34      def __len__(self):
35          return len(self.poses)
36
37      def get_pose(self, index): # fetches pose data for a given index
38
39          pose_data = self.poses.iloc[index]
40          if all(col in pose_data for col in ['eX', 'eY', 'eZ']):
41              # Convert Euler angles from degrees to radians
42              eul_x = np.deg2rad(pose_data['eX'])
43              eul_y = np.deg2rad(pose_data['eY'])
44              eul_z = np.deg2rad(pose_data['eZ'])
45              return {
46                  "trans_x": pose_data['tX'] /  1000.0,
47                  "trans_y": pose_data['tY'] / 1000.0,
48                  "trans_z": pose_data['tZ'] / 1000.0,
49                  "eul_x": eul_x,
50                  "eul_y": eul_y,
51                  "eul_z": eul_z,
52              }
53          else:
54              raise KeyError("Pose CSV does not contain expected rotation columns (eX
                    /eY/eZ)")
55
56  # Class for the dataset, integrating frames, depth maps, and poses
57  class Dataset_created:
58      def __init__(self, path_frames, path_depth_maps, path_poses):
59          self.frames = Frame_handler(path_frames)
60          self.depth_maps = Depth_map_handler(path_depth_maps)
61          self.poses = Pose_handler(path_poses)
62          self.length = min(len(self.frames), len(self.depth_maps), len(self.poses))
63
64      def __len__(self):
65          return self.length
66
67      def __getitem__(self, i):
```

```
68
69          return {
70              "Image": self.frames.get_frame(i),
71              "Depth_map": self.depth_maps.get_depth_map(i),
72              "Pose": self.poses.get_pose(i),
73          }
74
75      # To easily access all the GT depth maps
76      def get_all_depth_maps(self):
77          return [self.depth_maps.get_depth_map(i) for i in range(len(self))]
```

Listing C.3: Dataset handler classes for the created dataset.

C.2 Pre-processing Functions

This section presents the core pre-processing functions used to prepare depth maps from both the EndoSLAM UnityCam subset and the custom dataset. These routines handle unit conversion, masking, scaling, and formatting to ensure consistency before 3D reconstruction.

C.2.1 EndoSLAM Dataset

The pre-processing functions for the EndoSLAM dataset are organized according to their application. The function `prepare_data()` is used for single-frame pre-processing, including preparation for visualization and depth map comparison. In contrast, `prepare_gt_data()` and `prepare_predicted_data()` are used for full 3D reconstruction.

Auxiliary routines such as `apply_circ_mask()` and `resize_median_scalling()` are included at the end of this section, as they support both evaluation and reconstruction pipelines.

`prepare_data`

The `prepare_data()` function centralizes the pre-processing pipeline for single-frame evaluation of both GT and predicted depth maps. It performs inversion and unit conversion for GT maps, applies median scaling to predicted maps and normalizes both for visualization. A circular mask is also applied to remove border regions irrelevant to the endoscopic field of view and for a fair comparison.

```
1  def prepare_data(gt_depth_map, pred_depth_map):
2
3      gt_depth_map = gt_depth_map.max() - gt_depth_map  # Invert GT values
4      gt_depth_map = gt_depth_map / 100  # Convert from cm to meters
5      vis_gt_depth_map = norm(gt_depth_map)
6      gt_depth_map = apply_circ_mask(gt_depth_map)
7      comp_gt_depth_map = gt_depth_map  # For metric comparison
8
```

```
9
10      pred_depth_map = resize_median_scalling(gt_depth_map, pred_depth_map, eval_mono
            =True)
11      vis_pred_depth_map = norm(pred_depth_map)
12      pred_depth_map = apply_circ_mask(pred_depth_map) # Ignore irrelevant border
            pixels
13      comp_pred_depth_map = pred_depth_map
14
15      return gt_depth_map, pred_depth_map, vis_gt_depth_map, vis_pred_depth_map,
            comp_gt_depth_map, comp_pred_depth_map
```

Listing C.4: Preprocessing for depth map comparison and visualization.

prepare_predicted_data

This function is used for full 3D reconstruction from predicted depth maps. It resizes each map to a fixed resolution of 320×320 to match the RGB frame and ensure compatibility with the camera intrinsics. A circular mask is then applied to preserve only the valid endoscopic field of view.

```
1  def prepare_predicted_data(pred_depth_map, gt_depth_map):
2      pred_depth_map = resize_pred_depth(gt_depth_map, pred_depth_map)
3      pred_depth_map = apply_circ_mask(pred_depth_map)
4      return pred_depth_map
```

Listing C.5: Preprocessing for 3D reconstruction from predicted depth.

prepare_gt_data

This function is used to prepare the ground truth depth maps for full 3D reconstruction using the EndoSLAM dataset. It includes inversion and unit conversion, and optionally applies a circular mask to restrict the region to the valid endoscopic view.

```
1  def prepare_gt_data(gt_depth_map):
2      gt_depth_map = gt_depth_map.max() - gt_depth_map  # Invert depth values
3      gt_depth_map = gt_depth_map / 100  # Convert from cm to meters
4      gt_depth_map = apply_circ_mask(gt_depth_map)
5      #gt_depth_map = cv2.GaussianBlur(gt_depth_map, (5, 5), 0)  # Optional smoothing
6      return gt_depth_map
```

Listing C.6: GT depth map preparation for full 3D reconstruction.

resize_median_scalling

This function resizes the predicted depth map to match the GT resolution and applies median scaling, a common post-processing step in monocular depth estimation. Median scaling adjusts the predicted values to better align with the ground truth scale, particularly when the predicted depths are in a relative scale.

```
def resize_median_scalling(gt_depth, pred_depth, eval_mono=True):
    """
    Resize and scale the predicted depth map to align with ground truth.

    Args:
        gt_depth (HxW): Ground truth depth map.
        pred_depth (HxW): Predicted depth map.
        eval_mono (bool): If True, apply median scaling for monocular depth.

    Returns:
        resized_pred_depths: Scaled and resized predicted depth map.
    """
    if pred_depth.mean() != -1:
        gt_height, gt_width = gt_depth.shape[:2]

        # Resize prediction to GT resolution
        pred_depth = cv2.resize(pred_depth, (gt_width, gt_height))

        # Create mask for valid depth values within range
        mask = np.logical_and(gt_depth > 1e-3, gt_depth < 80)

        # Crop the central region to avoid border artifacts
        crop = np.array([
            0.40810811 * gt_height, 0.99189189 * gt_height,
            0.03594771 * gt_width,  0.96405229 * gt_width
        ]).astype(np.int32)
        crop_mask = np.zeros_like(mask, dtype=bool)
        crop_mask[crop[0]:crop[1], crop[2]:crop[3]] = True
        mask = np.logical_and(mask, crop_mask)

        val_pred_depth = pred_depth[mask]
        val_gt_depth = gt_depth[mask]

        # Apply median scaling
        ratio = 1
        if eval_mono:
            ratio = np.median(val_gt_depth) / np.median(val_pred_depth)
            val_pred_depth *= ratio

        resized_pred_depths = pred_depth * ratio
        return resized_pred_depths
```

Listing C.7: Resize and median scale predicted depth map.

apply_circ_mask

This utility function applies a circular binary mask to remove pixels outside the relevant endoscopic field of view.

```
def apply_circ_mask(pred_depth_map):
    pred_height, pred_width = pred_depth_map.shape
    yy, xx = np.meshgrid(np.arange(pred_width), np.arange(pred_height))
    center_x, center_y = pred_width // 2, pred_height // 2
    radius = int((min(pred_width, pred_height) // 2) * 1.08)
    circle_mask = (xx - center_x)**2 + (yy - center_y)**2 <= radius**2
    return pred_depth_map * circle_mask.astype(pred_depth_map.dtype)
```

Listing C.8: Circular mask application for field-of-view consistency.

C.2.2 Uterus Phantom Dataset

prepare_data_created_dataset

This function prepares both the ground truth and predicted depth maps from the uterus phantom dataset for visualization and point cloud generation. Unlike the EndoSLAM subset, no circular mask is applied, as this dataset does not replicate an endoscopic field of view. The GT maps are converted from millimeters to meters and clipped to discard background elements. The predicted maps are uncropped to match the original resolution.

```
def prepare_data_created_dataset(gt_depth_map, pred_depth_map):
    gt_depth_map = gt_depth_map / 1000  # Convert from mm to meters
    gt_depth_map[gt_depth_map > 0.6] = 0  # Focus on objects at table height
    vis_gt_depth_map = norm(gt_depth_map)

    pred_depth_map = uncropping_predicted_depth(pred_depth_map)
    vis_pred_depth_map = norm(pred_depth_map)

    return gt_depth_map, pred_depth_map, vis_gt_depth_map, vis_pred_depth_map
```

Listing C.9: Preprocessing for GT and predicted depth maps from the created dataset.

uncropping_predicted_depth

This helper function restores the predicted depth maps to their original 640×480 resolution. It fills the cropped horizontal region with zeros while preserving the predicted content in its appropriate spatial position.

```
def uncropping_predicted_depth(pred_cropped):
    crop_left_fraction = 0.43
    crop_right_fraction = 0.18
    H, W = (480, 640)

    crop_left = int(W * crop_left_fraction)
    crop_right = W - int(W * crop_right_fraction)

    expected_width = crop_right - crop_left
    expected_height = H

    if pred_cropped.shape != (expected_height, expected_width):
        pred_cropped = cv2.resize(pred_cropped, (expected_width, expected_height),
            interpolation=cv2.INTER_LINEAR)

    pred_full = np.zeros((H, W), dtype=pred_cropped.dtype)
    pred_full[:, crop_left:crop_right] = pred_cropped
    return pred_full
```

Listing C.10: Uncropping function to restore predicted map dimensions.

C.3 Camera Intrinsic Parameters

This section lists the intrinsic matrices used for projecting depth maps into 3D coordinates. Each matrix follows the pinhole camera model defined in Equation 3.8.

EndoSLAM Dataset (RGB camera)

$$K_{\text{EndoSLAM}} = \begin{bmatrix} 156.0418 & 0 & 178.5604 \\ 0 & 155.7529 & 181.8043 \\ 0 & 0 & 1 \end{bmatrix}$$

Uterus Phantom Dataset

RGB Camera (used for predicted depth):

$$K_{\text{RGB}} = \begin{bmatrix} 619.50 & 0 & 334.81 \\ 0 & 619.50 & 247.77 \\ 0 & 0 & 1 \end{bmatrix}$$

Depth Sensor (used for ground truth depth):

$$K_{\text{Depth}} = \begin{bmatrix} 381.39 & 0 & 324.98 \\ 0 & 381.39 & 239.34 \\ 0 & 0 & 1 \end{bmatrix}$$

C.4 Point Cloud Generation and Transformation Functions

This appendix contains the core functions used to convert depth maps into 3D point clouds and to generate transformation matrices from camera poses.

Pixel-to-Camera Projection

`pixel2cam()` transforms each depth value into a 3D point in the camera coordinate system using the inverse of the intrinsic matrix. It assumes the depth map is already in meters and filters out zero-valued (invalid) depth pixels.

```
def pixel2cam(depth, intrinsics_inv):
    rows, cols = depth.shape
    i_coords, j_coords = np.meshgrid(np.arange(rows), np.arange(cols), indexing='ij')
    depth_flat = depth.flatten()
    i_coords_flat = i_coords.flatten()
    j_coords_flat = j_coords.flatten()

    valid_mask = depth_flat != 0
    depth_valid = depth_flat[valid_mask]
    i_coords_valid = i_coords_flat[valid_mask]
    j_coords_valid = j_coords_flat[valid_mask]

    pixel_coords = np.vstack((j_coords_valid, i_coords_valid, np.ones_like(depth_valid)))
    cam_coords = intrinsics_inv @ pixel_coords
    cam_coords = cam_coords * depth_valid
    return cam_coords.T
```

Listing C.11: Project depth map to 3D camera coordinates.

Pose to Transformation Matrix Conversion

These functions convert camera pose representations (Euler angles or quaternions) into full 4×4 transformation matrices for use in world-coordinate projection.

`euler2mat()`, `quat2mat()`, and `pose_vec2mat()` are adapted from the original EndoSLAM implementation.

```
def euler2mat(angle):
    B = angle.size(0)
    x, y, z = angle[:, 0], angle[:, 1], angle[:, 2]

    cosz = torch.cos(z)
    sinz = torch.sin(z)

    zeros = z.detach()*0
    ones = zeros.detach()+1
    zmat = torch.stack([cosz, -sinz, zeros,
                        sinz,  cosz, zeros,
                        zeros, zeros,  ones], dim=1).reshape(B, 3, 3)

    cosy = torch.cos(y)
    siny = torch.sin(y)

    ymat = torch.stack([cosy, zeros,  siny,
                        zeros,  ones, zeros,
                        -siny, zeros,  cosy], dim=1).reshape(B, 3, 3)

    cosx = torch.cos(x)
    sinx = torch.sin(x)

    xmat = torch.stack([ones, zeros, zeros,
                        zeros,  cosx, -sinx,
                        zeros,  sinx,  cosx], dim=1).reshape(B, 3, 3)

    rotMat = xmat @ ymat @ zmat
    return rotMat
```

Listing C.12: Convert Euler angles to rotation matrix.

```
def quat2mat(quat):
    quat = quat / torch.norm(quat, dim=1, keepdim=True) # normalize the quaternion
    x, y, z, w = quat[:, 0], quat[:, 1], quat[:, 2], quat[:, 3] # extract
        components [x, y, z, w]

    # Compute the rotation matrix using the standard formula
    rot_mat = torch.stack([
        1 - 2 * (y ** 2 + z ** 2),     2 * x * y - 2 * z * w,         2 * x * z + 2
            * y * w,
        2 * x * y + 2 * z * w,         1 - 2 * (x ** 2 + z ** 2),     2 * y * z - 2
            * x * w,
        2 * x * z - 2 * y * w,         2 * y * z + 2 * x * w,         1 - 2 * (x **
            2 + y ** 2)
    ], dim=1).reshape(-1, 3, 3)

```

```
12      return rot_mat
```

Listing C.13: Convert quaternion to rotation matrix.

```
1  def pose_vec2mat(vec, rotation_mode='euler'):
2      translation = vec[:, :3].unsqueeze(-1)  # [B, 3, 1]
3      rot = vec[:, 3:]
4      if rotation_mode == 'euler':
5          rot_mat = euler2mat(rot)  # [B, 3, 3]
6      elif rotation_mode == 'quat':
7          rot_mat = quat2mat(rot)  # [B, 3, 3]
8      transform_mat = torch.cat([rot_mat, translation], dim=2)  # [B, 3, 4]
9
10     batch_size = transform_mat.shape[0]
11     bottom_row = torch.tensor([0, 0, 0, 1], dtype=transform_mat.dtype, device=
           transform_mat.device).view(1, 1, 4).repeat(batch_size, 1, 1)
12     transform_mat = torch.cat([transform_mat, bottom_row], dim=1)  # [B, 4, 4]
13
14     return transform_mat
```

Listing C.14: Build 4x4 transformation matrix from pose vector.

References

[1] S. Ali et al. 3d reconstruction from endoscopic images: A comprehensive survey. *Computer Methods and Programs in Biomedicine*, 2024.

[2] Cancer Council Australia. Polyps, 2025. Accessed May 2025.

[3] Costin Berceanu, Nicolae Cernea, Răzvan Grigoraș Căpitănescu, Alexandru Cristian Comănescu, Ștefan Paitici, Ioana Cristina Rotar, Roxana Elena Bohîlţea, and Maria Victoria Olinca. Endometrial polyps. *Romanian Journal of Morphology and Embryology*, 63(2):323–334, 2022.

[4] Blender Foundation. Blender – a 3D modelling and rendering package, 2024. Version 3.6. Accessed 2025-06-22.

[5] Taylor L. Bobrow, Mayank Golhar, Rohan Vijayan, Venkata S. Akshintala, Juan R. Garcia, and Nicholas J. Durr. Colonoscopy 3d video dataset with paired depth from 2d–3d registration. *Medical Image Analysis*, 73:102198, 2021.

[6] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, Jose Neira, Ian Reid, and John J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[7] John J. Craig. *Introduction to Robotics: Mechanics and Control*. Pearson Prentice Hall, 3rd edition, 2005.

[8] Jia Deng, Wei Dong, Richard Socher, Li Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2009*, pages 248–255. IEEE Computer Society, 2009.

[9] A. Di Spiezio Sardo et al. Hysteroscopy and treatment of uterine polyps. *Best Practice Research Clinical Obstetrics and Gynaecology*, 29:908–919, 2015.

[10] E. Dreisler, S. S. Sorensen, P. H. Ibsen, and G. Lose. Prevalence of endometrial polyps and abnormal uterine bleeding in a danish population aged 20-74 years. *Ultrasound in Obstetrics & Gynecology*, 34(6):634–639, 2009.

[11] Mark Hans Emanuel. New developments in hysteroscopy. *Best Practice & Research Clinical Obstetrics and Gynaecology*, 27:421–429, 2013.

[12] EndoSLAM Authors. Endoslam github repository, 2020. Accessed 2025-06-22.

[13] Grand View Research. Hysteroscopy procedures market size, share & trends analysis report, 2024. Accessed 2025-05-01.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[15] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. 4 2017.

[16] Natalia Ignaszak-Kaus, Karolina Chmaj-Wierzchowska, Adrian Nowak, Katarzyna Wszołek, and Maciej Wilczak. An overview of outpatient hysteroscopy. *Journal of Clinical and Experimental Obstetrics & Gynecology*, 2025.

[17] IHR India. Hysteroscopy, 2025. Accessed 2025-02-09.

[18] Intel Corporation. Intel realsense depth camera d435i, 2023. Accessed 2025-06-15.

[19] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1125–1134, 2017.

[20] Kenhub. Stomach histology, n.d. Accessed 2025-06-22.

[21] Keiji Kuroda, Mari Kitade, Iwaho Kikuchi, Jun Kumakiri, Shozo Matsuoka, Sachiko Tokita, Masako Kuroda, and Satoru Takeda. A new instrument: A flexible hysteroscope with narrow band imaging system—optical quality comparison between a flexible and a rigid hysteroscope. *Minimally Invasive Therapy & Allied Technologies*, 20(6):336–342, 2011.

[22] Trina Mansour and Yuvraj S. Chowdhury. Endometrial polyp, 2023. Last updated April 25, 2023. Accessed May 2025.

[23] mhdiksanprasetyo2003. Uterus, 2023. Model licensed under CC-BY-4.0. Accessed 2025-06-22.

[24] J.F. Moore and J. Carugno. Hysteroscopy, 2023. Updated 2023 Jul 18. Accessed May 2025.

[25] Wilson Gavião Neto, Jacob Scharcanski, Jan-Michael Frahm, and Marc Pollefeys. Hysteroscopy video summarization and browsing by estimating the physician's attention on video segments.

[26] Kutsev Bengisu Ozyoruk, Guliz Irem Gokceler, Taylor L. Bobrow, Gulfize Coskun, Kagan Incetan, Yasin Almalioglu, Faisal Mahmood, Eva Curto, Luis Perdigoto, Marina Oliveira, Hasan Sahin, Helder Araujo, Henrique Alexandrino, Nicholas J. Durr, Hunter B. Gilbert, and Mehmet Turan. Endoslam dataset and an unsupervised monocular visual odometry and depth estimation approach for endoscopic videos. *Medical Image Analysis*, 71, 7 2021.

[27] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. 8 2020.

[28] Anita Rau, Sophia Bano, Yueming Jin, Pablo Azagra, Javier Morlana, et al. Simcol3d—3d reconstruction during colonoscopy challenge. MICCAI Endoscopy Challenge, 2024. Accessed 2025.

[29] Anita Rau, P. J. Eddie Edwards, Omer F. Ahmad, Paul Riordan, Mirek Janatka, Laurence B. Lovat, and Danail Stoyanov. Implicit domain adaptation with conditional generative adversarial networks for depth prediction in endoscopy. *International Journal of Computer Assisted Radiology and Surgery*, 14(6):1097–1105, 2019.

[30] Alexander Richter, Till Steinmann, Jean Claude Rosenthal, and Stefan J. Rupitsch. Advances in real-time 3d reconstruction for medical endoscopy, 5 2024.

[31] Rais Shuaibu Muhammad Ibraheem Dauda Katagum Olubunmi Peter Ladipo Safiyya Faruk Usman, Efena Ross Efetie. The scope of hysteroscopy in the diagnosis and management of intrauterine conditions at a public fertility center in north-central nigeria: A retrospective study. *African Journal of Reproduction and Gynaecological Endoscopy*, 7:11–16, 2022.

[32] Christina Alicia Salazar and Keith B. Isaacson. Office operative hysteroscopy: An update. *Journal of Minimally Invasive Gynecology*, 25(2):199–208, 2018.

[33] Servier Medical Art. Servier medical art, 2024. Licensed under CC BY 3.0. Accessed 2025-06-24.

[34] Chrisostomos Sofoudis, Fani Grozou, Orestis Tsonis, and Minas Paschopoulos. See and treat hysteroscopy: Future challenges and new prospective. *Obstetrics & Gynecology International Journal*, 14(5), 2023.

[35] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. 9 2020.

[36] P. Tarneja and B. S. Duggal. Hysteroscopy: Past, present and future. *Medical Journal Armed Forces India*, 58(4):293–294, 2002.

[37] Nikolaos Tsampras, Kenneth Ma, Rohit Arora, Gemma McLeod, Flurina Minchelotti, and Laurentiu Craciunas. Office hysteroscopy safety and feasibility in women receiving anticoagulation and anti-platelet treatment. *European Journal of Obstetrics & Gynecology and Reproductive Biology*, 259:163–168, 2021.

[38] P. Tsonis, A. Petrou, D. Rontogianni, et al. The history of hysteroscopy as an endoscopic method. *International Journal of Gynecological Endoscopy*, 9(2):45–54, 2023. Accessed May 2025.

[39] G. Unfried, L. Pallwein, D. Gruber, C. Holler, A. Graf, C. Tempfer, L. Hefler, E. Hanzal, and H. Koelbl. Comparison of rigid and flexible hysteroscopes for outpatient hysteroscopy. *Fertility and Sterility*, 75(2):364–366, 2001.

[40] Salvatore G. Vitale, Andrea Giannini, Jose Carugno, Bruno van Herendael, Gaetano Riemma, Luis Alonso Pacheco, Amal Drizi, Liliana Mereu, Stefano Bettocchi, Stefano Angioni, and Sergio Haimovich. Hysteroscopy: where did we start, and where are we now? the compelling story of what many considered the "cinderella" of gynecological endoscopy. *Archives of Gynecology and Obstetrics*, 308(1):1–10, 2024.

[41] Lindsay Wells. Importance of endometrial polyp removal, 2023. Accessed 2025-03-01.

[42] Stephanie L. Wethington, Thomas J. Herzog, William M. Burke, Xuming Sun, Jodi P. Lerner, Sharyn N. Lewin, and Jason D. Wright. Risk and predictors of malignancy in women with endometrial polyps. *Annals of Surgical Oncology*, 18(13):3819–3823, 2011.

[43] Diana Wofk, Fangchang Ma, Tien-Ju Yang, Sertac Karaman, and Vivienne Sze. Fastdepth: Fast monocular depth estimation on embedded systems. 3 2019.

[44] Yuxiao Zhou and Kecheng Yang. Exploring tensorrt to improve real-time inference for deep learning.