VISUAL SLAM: SENSORS, EFFICIENCY, AND 3D OBJECTS

by

Jincheng Zhang

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Electrical Engineering

Charlotte

2024

Approved by:

Dr. Andrew R. Willis

Dr. James M. Conrad

Dr. Hamed Tabkhi

Dr. Srijan Das

ABSTRACT

JINCHENG ZHANG. Visual SLAM: Sensor, Efficiency, and 3D Objects. (Under the direction of DR. ANDREW R. WILLIS)

Simultaneous Localization and Mapping (SLAM) is pivotal for autonomous robots to navigate and operate in complex environments autonomously. Despite strides in SLAM technology, challenges persist in achieving accuracy and efficiency, especially in dynamic and resource-limited scenarios. This dissertation tackles three critical facets of the visual SLAM problem: environment simulation and sensor data processing, resource efficiency, and 3D object representation.

The first segment concentrates on simulating realistic environments and optimizing sensor data processing for mapping applications. It aims to enhance algorithm evaluation and refinement across diverse operational contexts, including UAV flights and challenging lighting conditions. The second part introduces a novel SLAM solution emphasizing low-bandwidth and computational constraints by capitalizing on planar semantic maps. Finally, the dissertation proposes an advanced method for 3D shape generation, integrating deep learning systems with shape grammars, to provide more accurate representations of common objects.

Contributions encompass a simulation framework tailored for mapping applications, a thermal sensor photometric correction model, an efficient RGB-D SLAM system emphasizing planar semantic mapping, and a fusion technique for enhanced 3D shape representation. These advancements collectively empower SLAM systems to perceive, navigate, and interact with spatial environments more effectively in the digital age. They enable agents to generate and communicate compressed map information within resource constraints, fostering closer collaboration between humans and robots.

ACKNOWLEDGEMENTS

I am deeply grateful to my advisor, Dr. Andrew Willis, for his exceptional guidance, support, and mentorship throughout my doctoral journey. His expertise, encouragement, and unwavering commitment to my success have been instrumental in shaping this dissertation and my academic growth.

I extend my sincere appreciation to my colleagues and peers for their collaboration, camaraderie, and intellectual exchange, which have enriched my research experience and broadened my perspectives.

I would like to thank my dissertation committee members, Dr. James Conrad, Dr. Hamed Tabkhi, and Dr. Srijan Das, for their valuable insights, feedback, and contributions to this work.

I am also grateful to the University of North Carolina at Charlotte and the Department of Electrical and Computer Engineering for providing the resources, facilities, and opportunities necessary for the completion of this dissertation. Additionally, I am grateful for the financial support provided by the Graduate Assistant Support Plan (GASP), which enabled me to pursue my doctoral studies.

To my family, I offer my deepest gratitude for their unwavering love, encouragement, and sacrifices throughout this endeavor. Their steadfast support has been my anchor, and I am truly blessed to have them by my side.

Lastly, I want to express my heartfelt appreciation to my girlfriend, Yuanyuan Gu. Her love, understanding, and encouragement have been my source of strength and inspiration. I am profoundly grateful for her unwavering support and presence in my life.

This dissertation is a testament to the collective support, encouragement, and contributions of all those mentioned above, and I am sincerely thankful for every one of them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

3D three-dimension

3DGANs 3D Generative Adversarial Network

3DSVAE 3D Shape Variational Autoencoder

AI Artificial Intelligence

BPS Bits per Symbol

CNN Convolutional Neural Network

DL Deep Learning

DSO Direct Sparse Odometry

DSOL Direct Sparse Odometry Lite

ECE Electrical and Computer Engineering

FOV Field Of View

GANs Generative Adversarial Networks

GNSS Global Navigation Satellite System

IMU Inertial Measurement Unit

IR Infrared

LZSS Lempel-Ziv-Storer-Szymanski

MAE Mean Absolute Error

mAP Mean Average Precision

mIoU Mean Intersection over Union

PSML Procedural Shape Modeling Language

QSS Quantization Step Size

RLE Run-Length Encoding

RMSE Root Mean Square Error

RMSE the Root Mean Square Error

SD the Standard Deviation

SDSO Stereo Direct Sparse Odometry

SfM Structure-from-Motion

SJC Score Jacobian Chaining

SLAM Simultaneous Location and Mapping

UAV Unmanned Aerial Vehicle

UNCC the University of North Carolina at Charlotte

VAE Variational Autoencoders

CHAPTER 1: INTRODUCTION

Simultaneous Localization and Mapping (SLAM) is a fundamental problem in robotics that involves building a map of an unknown environment while simultaneously tracking the robot's position. SLAM algorithms have become increasingly important in developing autonomous robots that can navigate and operate in unstructured environments without relying on pre-existing maps or GPS. The SLAM problem is challenging due to the need for robust sensor fusion and estimation techniques to overcome uncertainties and noise in sensor measurements. Over the years, various approaches to SLAM have been developed, ranging from probabilistic techniques like Kalman filters [1, 2, 3] to more recent deep learning-based approaches [4, 5, 6]. In this context, SLAM remains an active area of research and a critical technology in the advancement of robotics.

The emergence of modern consumer imaging sensors including RGB, depth, and infrared has had a significant impact on the visual SLAM research fields. They are low-cost, low-power, and low-size alternatives to traditional range sensors such as LiDAR [7]. Motivated by the advancement in imaging sensors, a lot of researchers have worked on visual SLAM (vSLAM), such as monocular SLAM [8, 9], RGB-D SLAM [10, 11], and stereo SLAM [12, 13].

1.1 Problem Statement

Despite the significant progress made in visual SLAM, existing SLAM approaches face several challenges that hinder their performance and applicability in real-world scenarios. This dissertation focuses on three perspectives of the visual SLAM problem: environment simulation and sensor data processing (Chapter 2), resource efficiency

(Chapter 3), and 3D object representation (Chapter 4). By delving into the intricacies of environment simulation and sensor data processing, this research endeavors to enhance the accuracy and robustness of SLAM systems in dynamic and complex environments. Furthermore, it explores methods to optimize resource utilization, ensuring that SLAM algorithms can operate efficiently across various platforms and computational constraints. Additionally, the dissertation delves into novel approaches for 3D object representation within SLAM frameworks, to improve scene understanding and facilitate higher-level tasks such as human-robot interaction.

1.1.1 Environment Simulation and Sensor Data Processing

Despite the advancements in simulation technologies [14, 15, 16] developed for evaluating SLAM systems, there remains a critical need to explore how simulations can effectively replicate real-world scenarios to conduct realistic experiments. While simulations offer the advantage of controlled environments and cost-effective experimentation, achieving high levels of realism that accurately represent complex real-world dynamics presents a significant challenge. Furthermore, there is a lack of standardized methodologies for assessing the fidelity and validity of simulated environments compared to their real-world counterparts. Addressing these challenges is essential to harnessing the full potential of simulations for conducting realistic experiments across various domains, ultimately advancing scientific research, technology development, and decision-making processes.

Conventional RGB-D cameras are popularly adopted as sensors in visual SLAM systems. However, their characteristic of operating under the human visible spectrum can be hindered by challenging environments such as fog, dust, and dynamic lighting conditions. This can cause visual SLAM to fail if distinct visual features are insufficiently available. This dissertation seeks to develop novel techniques that improve the fidelity of sensor data processing, thereby enhancing the overall performance of SLAM systems in challenging conditions.

1.1.2 Resource Intensity of SLAM Systems

Real-world applications of SLAM technology are limited by the robots' onboard resources. This is because the computational cost increases quadratically as map size grows [17], and the generated maps, popularly represented by point cloud, require significant memory and bandwidth budget to share with other robots. Researchers have attempted to mitigate these challenges by utilizing the map topology [18, 19] or reducing the problem size throughout the whole SLAM system including feature/frame selection, and keyframe/3D point decimation [20], at the expense of pose accuracy. Recent SLAM systems can utilize deep learning for extracting semantic information of the world [21, 22, 23, 24], however, such information is not used to simplify map representation, hindering downstream processing from saving computational cost and memory. These challenges make SLAM unsuitable for robots with limited resources, such as light-duty UAVs and swarm-style robots.

1.1.3 Shape Representations for 3D Objects

The generation and understanding of three-dimensional (3D) geometries hold significant importance across diverse domains, from computer graphics to robotics and virtual reality. Recent advancements in deep learning (DL) and generative modeling have propelled research in the area of 3D shape generation. Notably, techniques such as Variational Autoencoders (VAEs) [25, 26, 27], 3D Generative Adversarial Networks (3D-GANs) [28, 29, 30], and 3D Stable Diffusion [31, 32, 33, 34] have shown promise to autonomously produce realistic and diverse 3D shapes. Shape grammars have also been demonstrated as a powerful approach for formal model generation by providing a rule-based framework for generating complex geometric structures and enforcing constraints within objects [35, 36, 37]. Each methodology has its advantages and limitations. This research seeks to provide a fusion of these two methodologies to achieve the best of both worlds for novel 3D shape synthesis.

1.2 Structure and Contribution

To provide solutions to the aforementioned problems, this dissertation develops several novel techniques for SLAM systems. The contributions are divided into three topics: (1) environment simulation and sensor data processing for mapping applications, (2) SLAM with bounded computational cost and low bandwidth utilization, and (3) deep learning fusion with 3D procedural model representation. The remainder of Chapter 1 introduces the solutions this dissertation proposes to the three topics and summarizes the contributions of this dissertation. Chapters 2, 3, and 4 respectively further detail the three topics and the proposed approaches to addressed problems. For each topic a general overview of the problem and related literature is presented, followed by the proposed methodology which includes both completed work and proposed work, then concluded with an overview of how the dissertation makes a contribution to each of these fields. Chapter 5 concludes this dissertation and suggests further research.

Chapter 2 first presents an approach to simulating high-fidelity realistic environment and sensor data for SLAM applications. The approach was then used to analyze several mapping algorithms for UAV (Unmanned Aerial Vehicle) applications, focusing on low-altitude and high-speed scenarios. Findings quantify compromises in UAV algorithm selection, allowing researchers to find the mapping solution best suited to their application, which often requires a compromise between computational performance and the density and accuracy of geometric map estimates.

Chapter 2 then explores techniques that allow state-of-the-art SLAM systems to extend the concepts beyond the visible spectrum. A novel mathematical model that characterizes the pixel generation process of microbolometers, the infrared radiation detector, is proposed to solve the photometric correction problem for thermal cameras.

Chapter 3 proposes a semantic SLAM system that uses planar surfaces to greatly reduce the resource burden for localization and mapping. Two novel compression

algorithms for depth data and a method to independently fit planes to RGB-D data are provided so that plane data can be used for real-time odometry estimation and mapping. Additionally, the maps are extended with semantic information predicted from sparse geometries (planes) by a CNN.

Chapter 4 proposes a novel fusion of 3D shape representation using shape grammars and DL model estimation. Shapes are represented as a formal shape grammar using Procedural Shape Modeling Language (PSML) [38] which applies a sequence of rules to construct a 3D geometric model as a collection of 3D primitives. In contrast to competing approaches from the DL literature, the inclusion of dynamic parameterized formal shape models promises to allow DL applications to more accurately represent the structure of commonplace objects.

The contributions of this dissertation include:

1. An approach of simulating real-world environments and sensor data for the context of low-altitude high-speed UAV mapping application. (Chapter 2.1)
    (a) A novel approach to creating a real-world environment in simulation using the AirSim open-source simulator with Cesium Tiles Plugin that provides highly accurate 3D geometric models developed by Google.
    (b) A framework to simulate realistic flight for Unmanned Aerial Vehicles (UAVs).
    (c) An application of using the simulation framework to evaluate the different algorithms for UAV mapping.
2. An approach for state-of-the-art mapping solutions for RGB sensor data to be used for infrared sensors. (Chapter 2.2)
    (a) A photometric correction model for thermal sensors.

   (b) Experimental work showing the proposed photometric correction approach enables the infrared mapping results to achieve comparable performance with RGB mapping results.

3. An resource-efficient RGB-D SLAM system that constructs 3D planar semantic maps. (Chapter 3)

   (a) An efficient and effective compression algorithm for depth images.

   (b) A real-time fast plane fitting method that can fit planes independently of the sensor intrinsic camera parameters.

   (c) A real-time odometry algorithm based on plane constraints.

   (d) A CNN that performs semantic segmentation on plane input.

4. A novel fusion of deep learning systems and shape grammar for 3D shape generation. (Chapter 4)

   (a) Shape estimates can be guaranteed to satisfy complex geometric shapes and physical constraints.

   (b) Shape estimates are guaranteed to satisfy important geometric model properties by providing water-tight, i.e., manifold, and polygon models.

   (c) Shape estimates provide a highly compact parametric representation of objects allowing objects to be efficiently shared over communication links.

   (d) User-provided shape programs allow human-in-the-loop control over DL estimates.

   (e) Users can control the complexity and diversity of DL-estimated shapes for each object and each object component directly through the construction of the DL network.

(f) Object models can be used to synthesize training data for DL systems improving over current 3D model databases which use static 3D models and therefore lack geometric diversity.

(g) An example of the proposed DL fusion is provided that detects objects and their parametric representation given a PSML shape grammar is demonstrated.

These contributions significantly advance the state-of-the-art for SLAM. The environment simulation approach provides the ability to simulate or reproduce a real-world experiment in simulation and generate comparable results. The proposed photometric correction model for thermal infrared sensors equips the SLAM systems designed for RGB sensors to provide stable feature tracking and accurate mapping when processing thermal data. Taking advantage of the geometric primitives (planes and shape models generated using shape grammars), the proposed techniques provide new representations for 3D geometries that can (1) be quickly calculated and represent the data to a similar degree of geometric accuracy using far fewer parameters. The joint effect of these contributions allows agents with 3D sensing capabilities to calculate and communicate compressed map information commensurate with their onboard computational and bandwidth resources. Furthermore, the ability to interpret the world in a generative way that human beings also understand allows the possibility of closer collaboration between humans and robots.

REFERENCES

[1] S. Huang and G. Dissanayake, "Convergence and consistency analysis for extended kalman filter based slam," *IEEE Transactions on robotics*, vol. 23, no. 5, pp. 1036–1049, 2007.

[2] G. P. Huang, A. I. Mourikis, and S. I. Roumeliotis, "Analysis and improvement of the consistency of extended kalman filter based slam," in *2008 IEEE International Conference on Robotics and Automation*, pp. 473–479, IEEE, 2008.

[3] L. M. Paz, J. D. Tardós, and J. Neira, "Divide and conquer: Ekf slam in $o(n)$," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1107–1120, 2008.

[4] K. Tateno, F. Tombari, I. Laina, and N. Navab, "Cnn-slam: Real-time dense monocular slam with learned depth prediction," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6243–6252, 2017.

[5] C. Yu, Z. Liu, X.-J. Liu, F. Xie, Y. Yang, Q. Wei, and Q. Fei, "Ds-slam: A semantic visual slam towards dynamic environments," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1168–1174, IEEE, 2018.

[6] X. Long, W. Zhang, and B. Zhao, "Pspnet-slam: a semantic slam detect dynamic object by pyramid scene parsing network," *IEEE Access*, vol. 8, pp. 214685–214695, 2020.

[7] J. Civera and S. H. Lee, "Rgb-d odometry and slam," in *RGB-D Image Analysis and Processing*, pp. 117–144, Springer, 2019.

[8] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," September 2014.

[9] R. Mur-Artal and J. D. Tardos, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-d cameras," *IEEE Transactions on Robotics*, vol. 33, pp. 1255–1262, oct 2017.

[10] C. Kerl, J. Sturm, and D. Cremers, "Dense visual slam for rgb-d cameras," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2100–2106, IEEE, 2013.

[11] C. Wang, J. Yuan, and L. Xie, "Non-iterative slam," in *2017 18th International Conference on Advanced Robotics (ICAR)*, pp. 83–90, IEEE, 2017.

[12] J. Engel, J. Stückler, and D. Cremers, "Large-scale direct slam with stereo cameras," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1935–1942, IEEE, 2015.

[13] R. Wang, M. Schworer, and D. Cremers, "Stereo dso: Large-scale direct sparse visual odometry with stereo cameras," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3903–3911, 2017.

[14] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2149–2154, 2004.

[15] Robotis and Perception Group – University of Zurich, "Agilicious." https://github.com/uzh-rpg/agilicious. (accessed: Sep. 25, 2023).

[16] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, *Robot Operating System (ROS): The complete reference (volume 1)*, ch. 23, pp. 595–625. New York, NY, USA: Springer, Cham, 2016.

[17] Y. Park and S. Bae, "Keeping less is more: Point sparsification for visual slam," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7936–7943, IEEE, 2022.

[18] O. KÃ€hler, V. A. Prisacariu, and D. W. Murray, "Real-time large-scale dense 3d reconstruction with loop closure," in *Computer Vision – ECCV 2016*, pp. 500–516, Springer International Publishing, 2016.

[19] T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J. J. Leonard, and J. McDonald, "Real-time large scale dense rgb-d slam with volumetric fusion," *International Journal of Robotics Research: Special Issue on Robot Vision*, vol. 34, pp. 598 – 626, April 2015.

[20] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger, "Elasticfusion: Real-time dense SLAM and light source estimation," *The International Journal of Robotics Research*, vol. 35, no. 14, pp. 1697–1716, 2016.

[21] J. McCormac, A. Handa, A. Davison, and S. Leutenegger, "Semanticfusion: Dense 3d semantic mapping with convolutional neural networks," *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4628–4635, 2017.

[22] R. Dubé, A. Cramariuc, D. Dugas, H. Sommer, M. Dymczyk, J. Nieto, R. Siegwart, and C. Cadena, "Segmap: Segment-based mapping and localization using data-driven descriptors," *The International Journal of Robotics Research*, vol. 39, no. 2-3, pp. 339–355, 2020.

[23] S. Yang, Y. Huang, and S. Scherer, "Semantic 3d occupancy mapping through efficient high order crfs," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 590–597, IEEE, 2017.

[24] Z. Xuan and F. David, "Real-time voxel based 3d semantic mapping with a hand held rgb-d camera," 2018.

[25] E. A. Ajayi, K. M. Lim, S.-C. Chong, and C. P. Lee, "3d shape generation via variational autoencoder with signed distance function relativistic average generative adversarial network," *Applied Sciences*, vol. 13, no. 10, p. 5925, 2023.

[26] B. Dai and D. Wipf, "Diagnosing and enhancing vae models," *arXiv preprint arXiv:1903.05789*, 2019.

[27] A. R. Kosiorek, H. Strathmann, D. Zoran, P. Moreno, R. Schneider, S. Mokrá, and D. J. Rezende, "Nerf-vae: A geometry aware 3d scene generative model," in *International Conference on Machine Learning*, pp. 5742–5752, PMLR, 2021.

[28] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum, "Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling," *Advances in neural information processing systems*, vol. 29, 2016.

[29] A. Frühstück, N. Sarafianos, Y. Xu, P. Wonka, and T. Tung, "Vive3d: Viewpoint-independent video editing using 3d-aware gans," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4446–4455, 2023.

[30] E. R. Chan, C. Z. Lin, M. A. Chan, K. Nagano, B. Pan, S. De Mello, O. Gallo, L. J. Guibas, J. Tremblay, S. Khamis, *et al.*, "Efficient geometry-aware 3d generative adversarial networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 16123–16133, 2022.

[31] R. Liu, R. Wu, B. Van Hoorick, P. Tokmakov, S. Zakharov, and C. Vondrick, "Zero-1-to-3: Zero-shot one image to 3d object," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9298–9309, 2023.

[32] Stability AI, "3D couch generated using Zero123-XL models." https://stability.ai/news/stable-zero123-3d-generation. [accessed 07-Apr-2023].

[33] H. Wang, X. Du, J. Li, R. A. Yeh, and G. Shakhnarovich, "Score jacobian chaining: Lifting pretrained 2d diffusion models for 3d generation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12619–12629, 2023.

[34] J. Xu, X. Wang, W. Cheng, Y.-P. Cao, Y. Shan, X. Qie, and S. Gao, "Dream3d: Zero-shot text-to-3d synthesis using 3d shape prior and text-to-image diffusion models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 20908–20918, 2023.

[35] CityEngine, "http://www.esri.com/software/cityengine."

[36] L. Yang, J. Li, H.-T. Chang, Z. Zhao, H. Ma, and L. Zhou, "A generative urban space design method based on shape grammar and urban induction patterns," *Land*, vol. 12, no. 6, p. 1167, 2023.

[37] K. Zhang, N. Zhang, F. Quan, Y. Li, and S. Wang, "Digital form generation of heritages in historical district based on plan typology and shape grammar: case study on kulangsu islet," *Buildings*, vol. 13, no. 1, p. 229, 2023.

[38] A. R. Willis, P. Ganesh, K. Volle, J. Zhang, and K. Brink, "Volumetric procedural models for shape representation," *Graphics and Visual Computing*, vol. 4, p. 200018, 2021.

CHAPTER 2: SIMULATION AND SENSOR DATA PROCESSING

2.1 UAV-Borne Mapping Algorithms for Low-Altitude and High- Speed Drone Applications

2.1.1 Introduction

UAVs, also known as drones, have transcended conventional applications to become indispensable tools across an array of disciplines, from environmental monitoring and precision agriculture to disaster response and infrastructure inspection. At the heart of their efficacy lies the sophisticated interplay between UAVs and mapping algorithms, which serve as the backbone for converting raw sensor data into coherent, high-fidelity maps. These algorithms play a pivotal role in navigating complex terrains, extracting meaningful information, and ensuring precise localization of the UAV in real time. From traditional photogrammetry to advanced techniques like Simultaneous Localization and Mapping (SLAM), these algorithms continuously evolve to meet the diverse demands of UAV applications ranging from agriculture and forestry to disaster response and urban planning.

Mapping algorithms for UAVs are significantly influenced by flight altitude, dictating the scale of environmental perception and mapping capabilities. For high-altitude flights, the imagery changes between successive frames are slower than for low-altitude flights, which allows more overlap/correspondence between successive frames. However, as altitude increases, challenges such as reduced sensor performance, diminished feature visibility, and heightened geometric distortions emerge. While 3D or 2D laser scanners generate effective terrain models, their weight and sensitivity to ground proximity pose challenges. Compact depth-sensing devices, though commer-

cially available, often fall short in operational range. Camera-based mapping systems, while lightweight and scalable, face accuracy challenges at high altitudes due to reduced texture and discernible features. This limitation hampers feature tracking and matching, impacting overall mapping algorithm performance. High-altitude flights also amplify drift and uncertainty in UAV trajectory estimation, particularly affecting SLAM algorithms relying on sensor fusion. The accumulation of errors over time compromises poses estimations, emphasizing the critical consideration of flight altitude in optimizing mapping algorithm outcomes. This article focuses on multirotor UAVs and analyzes UAV algorithm performance at altitude ranges from 12 m to 20 m from the ground, which is considered to be "low-altitude" in this article. Investigations for this context provide an analysis of key sensor options and their strengths and weaknesses. Specific recommendations are also provided for light-duty UAVs (U.S. military UAS Group 1).

Mapping algorithms tailored for high-speed UAVs address the specific demands of dynamic and rapid flight scenarios. Real-time operation in these contexts is imperative, necessitating synchronization and integration of data from diverse sensors such as LiDAR, cameras, and inertial measurement units (IMUs). Adaptive navigation is equally crucial to accommodate the UAV's swift maneuvers and maintain mapping precision. Overcoming challenges related to large distances covered between sensor readings during high-speed flights is essential for achieving precise mapping results. Additionally, robustness in the face of environmental variability, including changes in lighting, weather conditions, and terrains, is vital. High-speed UAVs, integral in applications like surveillance and emergency response, benefit from ongoing advancements in mapping algorithms. These improvements enhance effectiveness, allowing UAVs to navigate rapidly changing environments and deliver precise and timely mapping outcomes. This article analyzes multirotor UAV algorithm performance for speed ranges from 15 m/s to 20 m/s which corresponds to the maximum speed for typical

commercially available platforms in this Group [1].

In recent decades, research investigating methods for 3D reconstruction from images has thrived. Examples of approaches include Structure-from-Motion (SfM) algorithms [2, 3, 4, 5, 6, 7] and stereo reconstruction (Stereo3D) algorithms [8, 9, 10, 11, 12]. These approaches are the algorithms that can be used as components of a SLAM system. SfM and Stereo3D differ in both computation methods and output formats. SfM algorithms analyze a sequence of 2D images from a camera and estimate the relative motion of the camera and the geometric structure of the observed 3D scene. Motion estimates include the camera pose, i.e., position and orientation, at each recorded image and the scene 3D structure observed in each image. Stereo3D estimates 3D scene structure from a pair of 2D images captured simultaneously by two cameras with known relative positions. Depth information is derived from the correspondence of observed scene points between the two images. While both SfM and Stereo3D target 3D scene reconstruction, they excel in different applications and scenarios.

The contributions of this article include:

- A comprehensive analysis outlining the strengths and limitations of state-of-the-art SfM and stereo reconstruction algorithms;
- A benchmark of the geometry accuracy and computation speed of various mapping algorithms;
- A theoretical foundation for sensor selection tailored to low-altitude and high-speed UAV mapping applications;
- A technical approach for extracting high fidelity geometric models from Cesium Tile data to perform analysis on 3D mapping and odometry algorithms;
- An innovative approach to simulate realistic flights, utilizing Unreal Engine for

high-realism environment synthesis, Cesium plug-in for geographical context, AirSim for vehicle dynamics, and PX4 Autopilot for precise vehicle control.

These contributions provide researchers new insight into how to best adopt mapping technologies for their UAV design in low-altitude and high-speed drone applications.

An initial discussion evaluates the theoretical suitability of a wide variety of sensors for this application and eliminates many sensors from candidacy for various technical reasons. Subsequent evaluation of algorithms is contingent on the proposed selection of best-practice sensors for this context.

Mapping algorithm analysis surveys current state-of-the-art real-time reconstruction algorithms suited to the sensors that were previously identified as appropriate for low-altitude and high-speed multirotor UAV mapping applications. From a wide array of possible algorithms, three were evaluated: (1) Direct Sparse Odometry (DSO) [13], (2) Stereo Direct Sparse Odometry (SDSO) [14], and (3) Direct Sparse Odometry Lite (DSOL) [15]. While many algorithms are available in the literature, the selected algorithms provide a representative sampling of reconstruction methods for the recommended camera sensors.

2.1.2 Related Work

This article compares three methods for 3D mapping in terms of their suitability for use on Group 1 UAVs at high-speed, low-altitude flight. Discussion of current sensing options indicates that camera-based methods are well-suited to this application. Experiments use a simulated environment to evaluate leading camera-based methods. For these reasons, a review of the related literature to this article is divided into three parts:

- A comparison of SfM and stereo3D reconstruction methods including recent leading implementations of these methods;
- A compact review of three 3D-from-images algorithms;

- A review of different 3D simulation options for developing and evaluating these mapping algorithms for the context of low-altitude high-speed flight.

A comprehensive literature review motivates the methodology and experimental approach for this article. Specifically, the choice of mapping algorithms analyzed and the simulation environment used was based on a comprehensive review of candidate solutions.

2.1.2.1 Structure-from-Motion vs. Stereo Reconstruction

Structure from Motion (SfM) and stereo reconstruction are two leading techniques employed in 3D reconstruction. This subsection describes the principles of both techniques to provide insights into their distinctive attributes and how they relate to high-speed low-altitude mapping applications.

Structure-from-Motion

SfM [16] is the process of reconstructing a 3D structure from its projections into a series of images taken from different viewpoints. It leverages the relative movement between a camera and objects in a scene to reconstruct the 3D structure. SfM estimates the camera poses and the spatial arrangement of points in the scene by analyzing the changes in perspective across multiple images. SfM has been extensively studied and applied in diverse fields, including 3D modeling [17, 18], augmented reality [19, 20], autonomous navigation [21, 22], and remote sensing [23, 24]. Researchers have explored various algorithms and optimization methods to enhance the accuracy [2, 25, 26, 27] and efficiency [4, 28, 6, 7] of SfM, making it a robust solution for scenarios where camera poses change dynamically, a common occurrence in high-speed low-altitude flights.

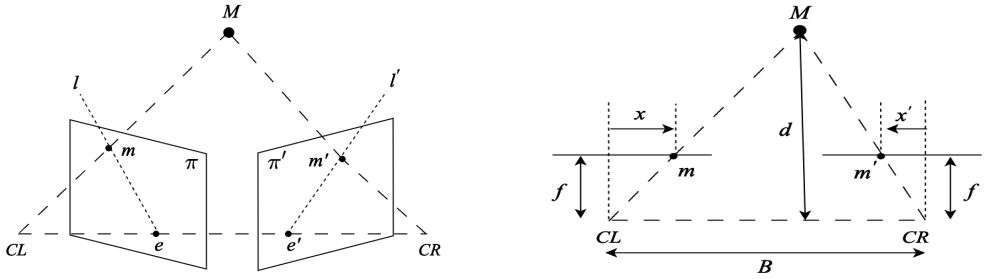
Stereo Reconstruction

Stereo reconstruction (stereo3D) involves the process of estimating the 3D structure of a scene from a pair of 2D images captured by two cameras with known relative

positions. By analyzing the disparities between the two images, stereo reconstruction algorithms can calculate the depth information of the scene points. This depth information allows for the creation of a 3D representation of the scene. Stereo3D is critical to enabling autonomous capabilities in a wide range of fields including robotics [29, 30], autonomous vehicles [31, 9], and 3D modeling [32, 33, 12].

Figure 2.1a illustrates the epipolar geometry of two pinhole cameras observing a 3D point **M**. Stereo reconstruction estimates **M**'s distance by analyzing its projections **m** and $\mathbf{m}'$. The baseline B connects camera origins **CL** and **CR**, defining epipolar geometry with epipoles **e** and $\mathbf{e}'$. The epipolar plane intersects with image planes $\boldsymbol{\pi}$ and $\boldsymbol{\pi}'$, forming epipolar lines. According to epipolar geometry, **m** in $\boldsymbol{\pi}'$ lies on epipolar line l'. Depth estimation involves finding corresponding points, simplified by image rectification in Figure 2.1b, ensuring **m** and $\mathbf{m}'$ align. Depth d is estimated through triangulation represented by Equation (2.1), considering column differences from **m** and $\mathbf{m}'$ to the center of the left and right images, baseline B, focal length f, and pixel width δ in the rectified image sensor.

$$d = \frac{Bf}{\delta\left(x - x'\right)} \tag{2.1}$$



(**a**) (**b**)

Figure 2.1: (**a**) Epipolar geometry of two cameras. (**b**) Epipolar geometry of a rectified image pair.

Figure 2.2 shows the theoretical dependency between the baseline parameter of a stereo camera pair and the accuracy of the depth estimates that the stereo sensor will

produce. Red lines show the depth deviations associated with a ±1 pixel error in the disparity. The plot shows that the disparity decreases as a square of the depth and error increases as a square of the depth.

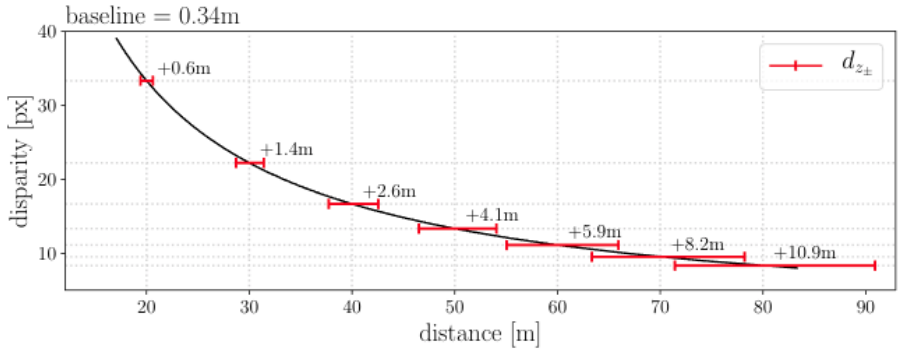


Figure 2.2: The dependency between depth estimation accuracy and the baseline of the stereo camera design for a baseline, B of 34 cm, based on [34].

SfM can be computationally intensive and requires feature matching and bundle adjustment for robust results. Stereo3D, with fixed camera positions, is typically less computationally intensive and more straightforward compared to SfM. SfM systems estimate the scene structure to an unknown scale and usually require fusion with other metric data, e.g., from an IMU or a GPS sensor, to make estimated geometric measurements consistent with the real geometric scene structure. Stereo3D directly estimates the scene structure and uses the baseline distance to provide scene scale estimates that are metrically consistent with the 3D scene geometry and do not require sensor fusion to recover the unknown scale.

2.1.2.2 Mapping Algorithms

This article focuses on the following three representative state-of-the-art real-time algorithms to investigate their applications to multirotor UAV-borne mapping:

- Structure-from-Motion: Direct Sparse Odometry (DSO) [13];
- Stereo Reconstruction: Stereo Direct Sparse Odometry (SDSO) [14] and Direct Sparse Odometry Lite (DSOL) [15].

DSO: Direct Sparse Odometry

DSO is a visual odometry technique that adapts SfM methods for 3D reconstruction. It directly estimates the camera motion and the sparse 3D structure of the environment from a sequence of 2D images by minimizing photometric errors. DSO differs significantly from traditional techniques by directly optimizing photometric errors in images, without relying on keypoint detectors or geometric priors. For a point, $\mathbf{p}$ in reference frame I_i, observed as $\mathbf{p}'$ in target frame I_j, the photometric error, given by Equation (2.2), is formulated as the weighted Sum of Squared Differences (SSD) over a small neighborhood of pixels.

$$E_{\mathbf{p}j} := \sum_{\mathbf{p}\in\mathcal{N}_{\mathbf{p}}} w_{\mathbf{p}} \left| 0\left(I_j\left[\mathbf{p}'\right] - b_j\right) - \frac{t_j e^{a_j}}{t_i e^{a_i}}\left(I_i[\mathbf{p}] - b_i\right) \right| 0_\gamma \tag{2.2}$$

where $\mathcal{N}_{\mathbf{p}}$ is the set of pixels in the SSD; (t_i, t_j) the exposure times of the frame I_i and I_j; (a_i, b_i, a_j, b_j) the brightness transfer variables defined in DSO for frame I_i and I_j, respectively, and $|0 \cdot |0_\gamma$ is the Huber norm. In addition to using robust Huber penalties, a gradient-dependent weighting $w_{\mathbf{p}}$ is applied. Further, $\mathbf{p}'$ stands for the projected point position of $\mathbf{p}$ with inverse depth $d_{\mathbf{p}}$, given by

$$\mathbf{p}' = \Pi_{\mathbf{c}}\left(\mathbf{R}\Pi_{\mathbf{c}}^{-1}\left(\mathbf{p}, d_{\mathbf{p}}\right) + \mathbf{t}\right) \tag{2.3}$$

with

$$\begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} := \mathbf{T}_j \mathbf{T}_i^{-1} \tag{2.4}$$

where $\Pi_{\mathbf{c}} : \mathbb{R}^3 \to \Omega$ denotes projection, $\Pi_c^{-1} : \Omega \times \mathbb{R} \to \mathbb{R}^3$ denotes back-projection, $\mathbf{c}$ denotes the intrinsic camera parameters, and $\mathbf{T}_i, \mathbf{T}_j \in \mathbf{SE}(3)$ are the camera poses represented by transformation matrices for frame I_i and I_j.

To minimize the photometric error between the corresponding points in two frames,

DSO incorporates a fully direct probabilistic model that jointly optimizes all model parameters, including camera motion and geometry, represented as inverse depth in a reference frame. The optimization is accomplished using the Gauss-Newton algorithm in a sliding window [35].

SDSO: Stereo Direct Sparse Odometry

SDSO is a stereo version of DSO. In a monocular mapping system like DSO, to initialize the whole system, i.e., to track the second frame with respect to the initial one using Equation (2.2), the inverse depth values $d_{\mathbf{p}}$ of the points in the first frame are required. In DSO, the points are initialized to have random depth values ranging from 0 to infinity, corresponding to a large depth variance. Unlike that, SDSO uses stereo matching to estimate a semi-dense depth map for the first frame, which significantly increases the tracking accuracy. The constraints from static stereo introduce scale information into the system. They also provide good geometric priors to temporal multi-view stereo.

DSOL: Direct Sparse Odometry Lite

DSOL presents an enhanced version of DSO and SDSO, proposing several algorithmic and implementation improvements to significantly speed up computation. Following the same practice as DSO of defining the photometric error in Equation (2.2), DSOL adopts the inverse compositional alignment method [36] to perform computationally expensive calculations, i.e., the Gauss-Newton approximation to the Hessian matrix, at the pre-computation phase, which largely improves the running speed of the algorithm. Compared to DSO and Stereo DSO, key aspects of optimization in DSOL include the following: (1) utilizing an inverse compositional alignment method for frame tracking, improving accuracy and speed; (2) adapting a better stereo photometric bundle adjustment formulation compared to SDSO; (3) simplifying keyframe creation and removal criteria from DSO, allowing for better utilization of computa-

tional resources and parallel processing; and (4) implementing algorithmic enhancements to streamline the computation process, making it more suitable for real-time applications, especially in resource-constrained environments. The focus of DSOL is on mapping speed and efficiency while maintaining accuracy.

2.1.2.3 Aerial Simulation Solutions

There are various simulation platforms for vehicles and environments catering to the diverse needs of researchers. Gazebo [37], with its open-source nature, stands as a versatile choice, emphasizing realism and adaptability. Agilicious [38] specializes in agile quadrotor flight, providing unique applications such as drone racing. RotorS [39], integrated with the Robot Operating System (ROS), offers high-fidelity UAV simulation. Flightmare [40], part of the AirSim project, excels in simulating multiple drones for swarm robotics research. Kumar Robotics Autonomous Flight [41] addresses GPS-denied quadcopter autonomy. MIT's FlightGoggles [42] offers an immersive experience with photorealistic graphics. AirSim, developed by Microsoft, on top of the Unreal Engine, excels in generating highly realistic perceptual simulation data in complex and dynamic environments.

An approach is proposed in [43] to reproduce real-world experiments in simulation using the AirSim open-source simulator with the Cesium Tiles plugin, allowing for large-scale 3D geometry analysis. This paper adapts the methodology and extends it with other aerial vehicle control technologies, achieving precise vehicle control in high-realism virtual models that replicate real-world contexts world.

2.1.3 Methodology

The overall approach for the methods of this article consists of three steps:

- Describe the benefits and shortcomings of various candidate sensing modalities for low-altitude high-speed mapping using Group 1 UAVs resulting in a recommendation for using one or more high-frame rate conventional camera sensors

for this application (Section 2.1.3.1).

- Describe the simulation methods used to collect data using a highly realistic 3D environment made possible by integrating the AirSim simulator with Google's 3D map database using the Cesium Tiles plugin for the Unreal Engine (Section 2.1.3.2).

- Describe the evaluation methods adopted to compare the mapping results generated from experimental flights within the simulated environment (Section 2.1.3.3).

2.1.3.1 Sensors for UAV Mapping

Three prominent sensor types are investigated as potential components of the UAV perceptual payload. These sensor types are listed below:

- LiDAR (Light Distance and Ranging) Sensors;

- Event Cameras;

- Conventional EO and IR Cameras.

Our assessment considered leading examples of each sensor that would be potentially appropriate for the high-speed low-altitude context and commercially available. The specifications of the sensors were then reviewed in terms of their ability to provide measurements that meet the requirements of UAV mapping. Based on this analysis, a determination was reached regarding the suitability of each sensor.

LiDAR

Figure 2.3 shows several LiDAR sensors evaluated for inclusion in the platform payload. LiDAR sensors have emerged as a popular choice for UAV mapping applications with significant advancements in LiDAR-based techniques [44, 45, 46, 47]. However, it was quickly determined that these devices would not be appropriate for

the UAV mapping application. The shortcomings of these sensors are described in the list below:

- Weight: LiDAR sensors typically weigh 500 g. or more which would be equivalent to approximately 5 image sensors of 100 g.
- Measurement Method: LiDAR sensors measure individual 3D points at one time or a collection of 3D points using a laser line-scanning technology. In either case, a rotating mirror in the sensor scans the scene over time. Accurate integration of scan data requires motion compensation for individual 3D point measurements for mapping and geometry estimation.
- Measurement Speed: LiDAR sensors typically scan at low rates (10–20 Hz) which makes the capture of a complete 3D scene geometry impractical for the rates required by high-speed flight.

(**a**) (**b**) (**c**) (**d**)

Figure 2.3: Several LiDAR sensors were evaluated for inclusion on the platform. Left to right are shown (**a**) the Ouster OS1, (**b**) the HRL131, (**c**) the RIEGL miniVUX-HA, and (**d**) the L3 Harris Tactical Geiger-Mode LiDAR sensors.

The data stream, resulting from the combination of the measurement method and measurement speed, requires highly accurate flight pose tracking over long distances at high speeds for the accurate integration of data into a unified 3D map. Achieving this may pose challenges considering the tracking accuracy limitations of onboard instruments, and substantial computation may be required for per-point or per-scan line motion compensation. Due to these reasons, the utilization of LiDAR sensing instrumentation for high-speed UAV mapping is not advisable.

Event Cameras

Figure 2.4 shows several event camera sensors evaluated for inclusion in the platform payload. The key attractive aspect of event cameras that has sparked considerable interest from researchers and industry alike is the extremely high temporal accuracy. Specifically, event cameras can resolve intensity changes in the perceptual field at a temporal resolution of approximately 1 μs. For this reason, event cameras have been used in high-speed contexts.

	DAVIS346 Simultaneous events and frames	**DVXplorer Lite** Discover event-based vision	**DVXplorer** High resolution	**DVXplorer Mini** Lightweight and compact	**DVXplorer S Duo** Smart camera
Event output					
Spatial resolution	346 x 260	320 x 240	640 x 480	640 x 480	640 x 480
Temporal resolution	1 µs (output precision)	65--200 µs (effective accuracy, full event frame)			
Max. throughput	12 MEPS	100 MEPS	165 MEPS	450 MEPS	450 MEPS
Typical latency	<1 ms	<1 ms	<1 ms	<1 ms	<1 ms

Figure 2.4: A collection of event cameras commercially available from the iniVation Corp [48].

While the deployment of event cameras as a component of UAVs may seem attractive due to the temporal resolution, there are several shortcomings associated with integrating this hardware into the UAV payload:

- Resolution: Resolution is a key parameter for depth accuracy as discussed in the stereo reconstruction Section 2.1.2.1. Accuracy strongly ties to both resolution and pixel size, δ, as shown in Figures 2.1b and 2.2. Event camera resolution, 0.3 MPixels, is a factor of 5–10 times lower than conventional image sensors, and the pixel size of $\delta = 18$ μm is a factor of 6–18 times larger than conventional image sensors, e.g., the Sony IMX472 sensor has a resolution of 21 megapixels

and a pixel size of 3.3 μm.

- Weight: While these sensors are lighter than LiDAR sensors, they weigh $\sim$100 g. and much lighter camera sensors are available.
- Latency: While the temporal resolution of event cameras is an impressive 1 μs, the latency of the measurements is on the order of $<$1 ms. This latency is similar to that of high frame rate conventional image sensors with frame rates of $+$100 fps and similar $<$1 ms latency.
- Nighttime Performance: Event cameras operate on similar principles to conventional visible light cameras. As such, they are suited to deployment in daytime contexts. The lack of an infra-red event camera requires completely separate perceptual software stacks for the vehicle in daytime and nighttime contexts.

Event cameras are a recent technology that has emerged and matured over the past decade. These sensors have unparalleled temporal resolution of 1 μs which makes them popular for capturing high-speed phenomena endemic to high vehicle speed applications. Yet, current technology has not matured to the extent required to make this sensor a viable option. Further, the development of a nighttime IR sensing event camera is an active area of sensor development under initiatives with no commercially viable examples. The drawbacks of having low sensor resolution, large pixel size, and no nighttime performance combined with comparable latency and weight to standard conventional cameras suggest that this sensor is not appropriate for inclusion as a component of the UAV payload for high-speed mapping applications.

Electro-Optical and Infrared Cameras

Conventional image sensors, including electro-optical (EO) and infrared (IR) sensors, have many beneficial attributes that often make them the sensor of choice for perception designs that must satisfy low Size, Weight, and Power (SWaP) requirements. They are well suited for UAV mapping tasks for several reasons:

- SWaP: Both EO and IR camera modules are available commercially in a very large variety of form factors. This includes a compact 25 mm^3 weighing 10–50 g requiring $\sim$1 W for power and providing temporally synchronized high framerate (60 fps) images.

- High-Quality Imaging: Modern cameras offer high-resolution imaging with the ability to capture fine details, which is crucial for mapping tasks, especially in scenarios where identifying objects is essential.

- Mapping and Geospatial Data: Cameras can be used for aerial imaging and photogrammetry to create detailed maps and 3D models of areas, making them valuable for urban planning, environmental monitoring, and disaster management.

- Stereo Vision: Cameras can be paired to create a stereo vision system. By capturing images from two slightly offset viewpoints, they can calculate depth information through triangulation, using the disparity between corresponding points in the two images. This method provides accurate 3D information.

- Integration with Other Technologies: Cameras can be integrated with other sensors and technologies, such as Inertial Measurement Units (IMUs) and Global Navigation Satellite System (GNSS), to enhance their capabilities and improve accuracy.

- Wide Field of View: Many cameras have wide-angle lenses or the ability to pan, tilt, and zoom (PTZ), providing a broad field of view and the flexibility to focus on specific areas of interest.

- Daytime and Nighttime Versatility: Camera sensors are capable of sensing in both daylight and nighttime conditions. If high sensitivity is needed in both scenarios, it is possible to replace daytime image sensors with infrared image

sensors during nighttime conditions. This can be achieved with minor modifications to the underlying software and algorithms.

- Large Active Algorithm Ecosystem: Researchers worldwide develop cutting-edge algorithms for these sensors at top institutions. Utilizing this sensor type enables leveraging the latest, optimized, and theoretically advanced algorithms for vehicle perception tasks.

- Cost-Effectiveness: Compared to some other sensing technologies, cameras are cost-effective, making them accessible for a wide range of surveillance and mapping applications.

Conventional image sensors have many beneficial attributes that make these sensors attractive for multirotor UAV applications. These sensors have low SWaP requirements and can record >16 M measurements at a time from the environment. These sensors can be combined with lens components that provide both wide-angle viewpoints, e.g., a 230° FOV via the fisheye lens, for omnidirectional perception and confined viewpoints, e.g., 80° FOV "standard" lens, for high fidelity target tracking and mapping. The intensive work required to integrate these sensors and develop optimized algorithms to process their data to work using onboard computing resources can be reused between daytime (EO) and nighttime (IR) sensing contexts.

Image sensors designed for both infrared (IR) and visible light often share common image processing algorithms, including basic processes like filtering, noise reduction, contrast enhancement, and image registration. Additionally, object detection, recognition, feature extraction, and image fusion algorithms can typically be adapted for both IR and visible light images, leveraging shared features and patterns. However, notable differences emerge, primarily related to spectral characteristics, illumination, noise, calibration, temperature considerations, environmental conditions, and the unique sensitivities of IR images to object materials. These distinctions

necessitate adjustments in algorithms to address variations in contrast, object recognition, and material discrimination, showcasing the need for specialized approaches in certain contexts.

Recommendations

The assessment of available sensors suggests that the conventional image sensors are the best-practice sensors for multirotor UAV applications. EO and IR sensors, being lighter in weight and faster in measurement speeds compared to LiDAR sensors, also offer better image quality and nighttime measurement capability in contrast to event cameras. The choice of conventional sensors not only aligns with budgetary constraints but also caters to the diverse needs of UAV operations, encompassing navigation, mapping, and surveillance with exceptional performance and reliability.

2.1.3.2 Benchmark Dataset

A virtual environment that mimics real-world scenes was used for evaluation. Compared to real datasets, synthetic datasets for evaluating mapping algorithms bring a notable advantage in the form of readily available ground truth 3D models. This availability of ground truth data facilitates a more rigorous assessment of mapping performance, ensuring precise comparisons between the algorithm's outputs and the known true state of the environment.

Environment Simulation

AirSim [49] was used to simulate the dynamics of the drone. AirSim, developed by Microsoft, stands as a groundbreaking and influential simulator that has become a cornerstone in the development of autonomous drones and robotics. What sets AirSim apart is its capacity to simulate complex and dynamic environments with exceptional fidelity, replicating not only the physics of flight but also the intricacies of various sensors like cameras (RGB and depth), LiDAR, and GPS. Figure 2.5 shows an example of the AirSim simulated images captured by the cameras mounted on

the multirotor.

(**a**) (**b**) (**c**) (**d**)

Figure 2.5: An example of an AirSim city environment showing the following: (**a**) the FPV view in the simulator where the drone is hovering, (**b**) RGB image from the simulated left camera mounted on the drone, (**c**) RGB image from the simulated right camera, and (**d**) depth image from the simulated depth sensor where objects closer to the depth camera appear darker.

The proposed approach used the Cesium plugin for the Unreal Engine, also known as "Unreal Cesium", to simulate real-world scenes, and enhance the effectiveness of simulations. Although AirSim provides rich virtual environments for testing and fine-tuning a wide array of autonomous systems, these environments are often designed for games and lack realism. To synthesize virtual models that replicate real-world contexts, AirSim can be integrated into the Unreal Engine to allow the Unreal Cesium plugin to create digital twins of real-world environment models. The Cesium plugin, given the latitude and longitude coordinates of desired locations, can load 3D tilesets at the location from Google Maps in the AirSim simulator.

The Unreal Cesium plugin creates a powerful combination by integrating the Unreal Engine's advanced rendering and simulation capabilities with Cesium's geo-spatial visualization and data streaming features. By streaming high-resolution 3D models from Google Maps, overlying them onto real-world maps, and applying dynamic lighting and shadows, to provide precise representations of real-world locations, such as cities, terrains, and 3D models of buildings. This integration allows developers to create highly realistic and spatially accurate virtual environments for various applications. Once the 3D map is generated, it behaves as a collision object in the Unreal Engine. The UAV then interacts with this model using the geometry of the

environment and a physics engine. Figure 2.6 demonstrates the benefits of the Unreal Cesium plugin for simulation. It shows three real-world locations: (1) the UNC Charlotte campus, USA, (2) the Grand Canyon, USA, and (3) Paris, France.

(**a**) (**b**) (**c**)

Figure 2.6: Realistic environments created using Unreal Engine and Cesium Plug-in. (**a**) UNC Charlotte, NC, USA; (**b**) Grand Canyon, AZ, USA; (**c**) Eiffel Tower, Paris, France.

Flight Simulation

Figure 2.7 depicts the proposed pipeline for flight simulation. QGroundControl and PX4-Autopilot are software components commonly used in the field of UAVs and drones. They work together to provide a comprehensive solution for controlling and managing drone flights. The missions are planned by defining waypoints, flight paths, and specific actions for the drone to perform and QGroundControl sends the mission plans to the autopilot system PX4-Autopilot. As an open-source flight control software for UAVs, PX4-Autopilot runs on the flight controller onboard the drone and is responsible for stabilizing the aircraft, executing flight plans, and interfacing with sensors and actuators. Controlled by PX4-Autopilot, the simulated UAV in AirSim follows the planned trajectory in a high-realism virtual environment created by Unreal Engine and Cesium plug-in. The cameras mounted on the UAV then capture the images (RGB, depth, etc) of the scene. These images, along with the ground truth vehicle odometry, can be obtained from AirSim, which then can be applied together to generate the ground truth 3D model of the world.

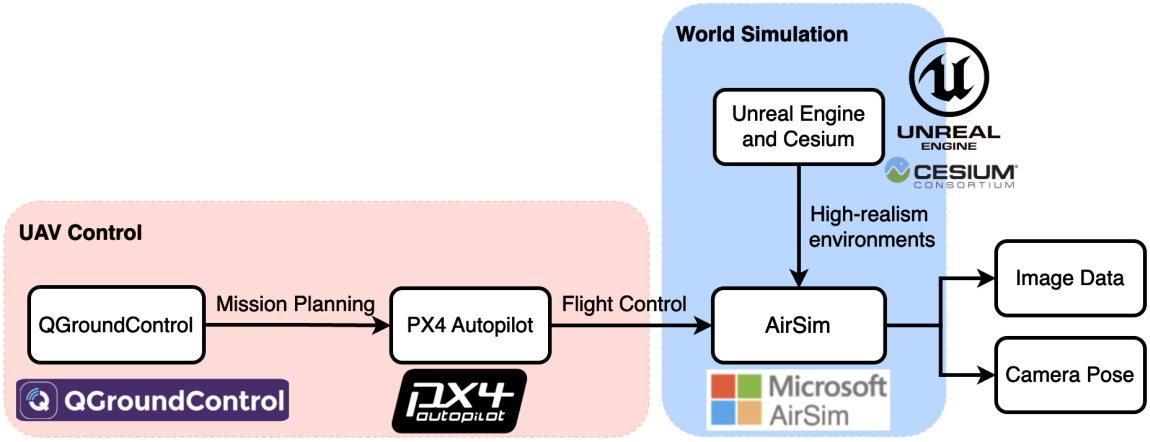


Figure 2.7: The flight simulation pipeline integrates the following four robot development technologies to facilitate development and testing: (1) Unreal Engine and Cesium plugin (high-realism image synthesis), (2) AirSim (vehicle dynamics), (3) QGroundControl (mission planning), and (4) PX4-Autopilot (vehicle control and Software-In-The-Loop).

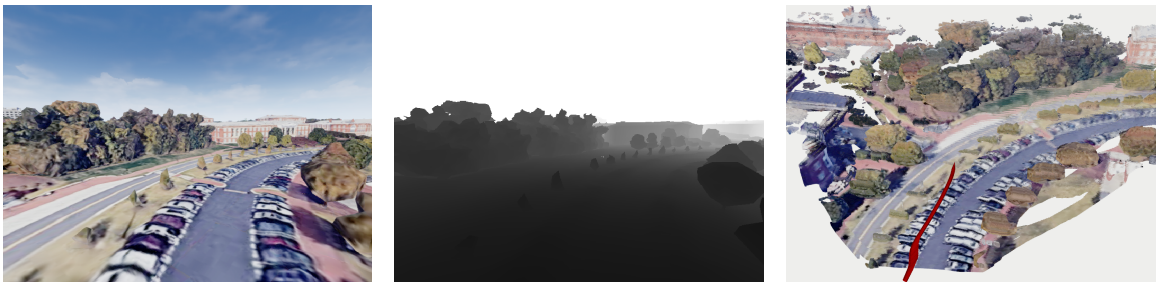
Ground Truth Geometry

The ground truth geometry is generated by applying AirSim's built-in functionality for ground truth pose and noiseless telemetry to collect color-attributed point cloud data from simulated noiseless pixel-aligned RGB and depth images captured by a drone that traverses the environment. The telemetry is extracted from Cesium Tile data for generating high-fidelity geometric models. Poses and point clouds are integrated using standard mapping methods to reconstruct the scene geometry. Figure 2.8 shows an example of integrating the drone odometry (shown as the red curve Figure 2.8c) and the point cloud from RGB-D image sequences to create a geometric model of the scene.

2.1.3.3 Evaluation Methods

Mapping algorithm performance is evaluated using the following two key performance criteria: (1) mapping accuracy, and (2) mapping speed. Mapping accuracy is assessed by comparing the geometry of the reconstructed point cloud with the ground truth point cloud. Geometric accuracy measures each mapping algorithm's ability to

faithfully capture spatial relationships in the environment. Algorithm performance speed quantifies the amount of 3D estimates generated per unit of allocated computational resources. More computation allows more points to be tracked in sequential frames and the creation of more keyframes. Both tracked points and keyframe data feed non-linear bundle adjustment and batch trajectory optimization processes which improve map fidelity but can require significant computational resources. The evaluation methods allow result analysis that indicates design trade-offs associated with each mapping algorithm.



(**a**) (**b**) (**c**)

Figure 2.8: Ground truth geometry can be generated by transforming the point clouds of RGB-D frame sequences to the odometry of the drone which is shown in red in Figure 2.8(c). (**a**) An RGB frame captured by the drone; (**b**) A depth frame captured by the drone; (**c**) Integrating pose and point clouds to generate a map.

Point Cloud Registration

Point cloud registration seeks to compute the alignment between two 3D point clouds measured from the same surfaces in distinct coordinate systems. Alignment algorithms identify point correspondences between the misaligned point cloud datasets and compute the rigid Euclidean transformation that makes corresponding points coincide. To accomplish this, a point cloud is selected as the staticdataset and all other measured point clouds are transformed to align with the measurement coordinate system of the static dataset. Figure 2.9 shows an example of registering two point clouds where the red point cloud is the source set and the blue is the static dataset.

The Iterative Closest Point (ICP) algorithm [50] is employed to estimate point

cloud alignments. The ICP algorithm consists of the following two steps: (1) compute correspondences and (2) compute the best alignment given the correspondence. Steps (1) and (2) are iterated until the alignment of Step (2) stops changing. Correspondences for a given iteration are calculated by finding the closest point in the static dataset to each point in the dataset being aligned. Closest point searches stop at a user-specified search radius for each point. The ICP algorithm seeks to minimize the RMSE (Root Mean Square Error) of all the distances between corresponding points and terminates when the gradient of RMSE is below a predefined threshold or a predetermined maximum iteration count is reached. Alignments resulting from the ICP algorithm are used to evaluate the geometry accuracy of mapping algorithms.

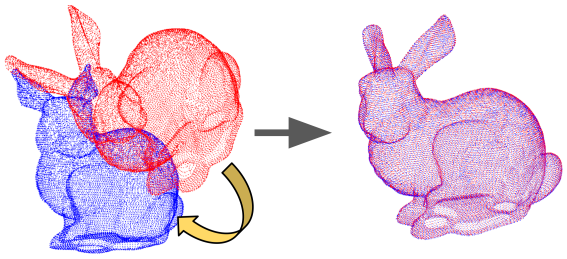


Figure 2.9: An example of point cloud registration [51]. Red: source point cloud. Blue: target point cloud. Purple: registration result.

Geometric Accuracy

With the correspondences found using the ICP algorithm, the geometric accuracy of the reconstructed map is measured by the distance between corresponding points in the ground truth 3D model and the reconstructed 3D map. The mean of this distance of all the corresponding points is used to evaluate the accuracy performance, calculated as follows:

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} ||\mathbf{P}_i - \mathbf{T}_i|| \tag{2.5}$$

where N is the total number of corresponding points, $\mathbf{P}_i$ is the position of the i-th corresponding point, and $\mathbf{T}_i$ is the ground truth (reference) position for that point.

Further, the standard deviation (std) of the errors (distances) is used to evaluate

the variability in the errors, calculated as follows:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(||\mathbf{P}_i - \mathbf{T}_i|| - \bar{x})^2}{N}} \tag{2.6}$$

Computational Cost

To assess the computational efficiency of mapping algorithms, two key metrics were focused on in this article: keyframe creation time and frame tracking time. These metrics were selected to provide insights into the mapping speed of the algorithms.

Keyframe Creation Time: Keyframe creation time quantifies the time required to identify keyframes during the mapping process. Keyframe creation is arguably the most time-consuming process of the mapping pipeline, often 5–10× slower than tracking [15]. Creating too many keyframes will cause the system to eventually lag behind the frame rate. Keyframe creation time reflects the computational efficiency of map reconstruction.

Frame Tracking Time: Frame tracking time represents the duration required for the algorithms to process and track individual frames with respect to the keyframes. This metric reflects the algorithm's ability to track and update the mapping information in real-time.

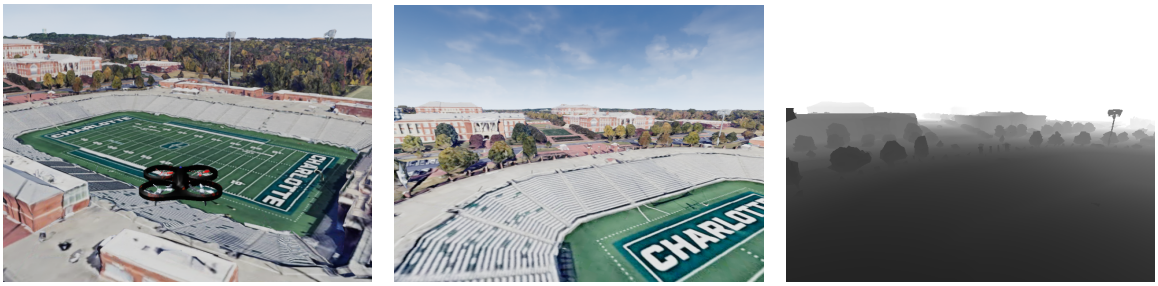
These two metrics collectively provide a comprehensive evaluation of the computational cost of mapping algorithms. The results of these evaluations are discussed in Section 2.1.4.2, providing the relative efficiency and performance trade-offs among the implemented algorithms.

2.1.4 Results

The experimental scene was a virtual model of the UNC Charlotte campus near the football stadium. The model was generated using AirSim and the underlying Unreal Engine in combination with the Cesium Tiles plugin.

Experiments were conducted on an Intel i7-12700KF CPU. The implementations of DSO and DSOL that were made available on GitHub by the authors were used [52, 53]. SDSO implementation by the authors is not available so an open-source third-party implementation on GitHub was chosen [54]. All implementations adhered to the original configuration optimized by their authors for accuracy and/or speed performance including the number of active keyframes and maximal tracking points per frame. Customized modifications made to all three algorithms respectively for collecting experimental data include the following: (1) saving the generated point cloud to a PCD file; (2) saving the keyframe ID and associated creation time to a text file; (3) saving the frame ID and associated tracking time to a text file.

Experiments consist of a simulated quadrotor vehicle that traverses the virtual scene at heights ranging from 12 m to 20 m and at speeds ranging from 16.5 m/s to 20 m/s. During the flight camera sensor telemetry was recorded from a stereo pair of camera mounts to the UAV chassis. Algorithms processed the telemetry to generate mapping data for the environment. The SfM algorithm (DSO) used data from the left camera of the stereo rig while DSOL and SDSO (stereo reconstruction) utilized all the available image data. The left camera is chosen to define the sensor coordinate system and the noiseless depth sensor is co-located with the left camera to record ground truth depth for each pixel measured within the view of the left camera. Figure 2.10a illustrates the simulated drone's flight over the UNC Charlotte football stadium, covering a 3-minute flight duration and capturing 1027 RGB stereo pair frames and associate ground truth depth. Sample images from the drone's simulated RGB and depth sensors are displayed in Figure 2.10b,c.



(**a**) (**b**) (**c**)

Figure 2.10: A simulated UNC Charlotte campus world. (**a**) A quadrotor flying in a virtual model of UNC Charlotte; (**b**) A RGB image captured by the drone camera; (**c**) A depth image captured by the drone camera .

2.1.4.1 Mapping Accuracy Evaluation

Using the methods of Section 2.1.3.2, a ground truth point cloud of the experimental scene was calculated which is shown in Figure 2.11. This point cloud serves as the ground truth geometry to evaluate the mapping accuracy of different algorithms. Figure 2.12 shows the point cloud respectively reconstructed by DSO, SDSO, and DSOL. All three maps qualitatively encode the shape and size of objects from the experimental scene. However, the point density and scene details of DSO and DSOL maps outperform the DSOL map.

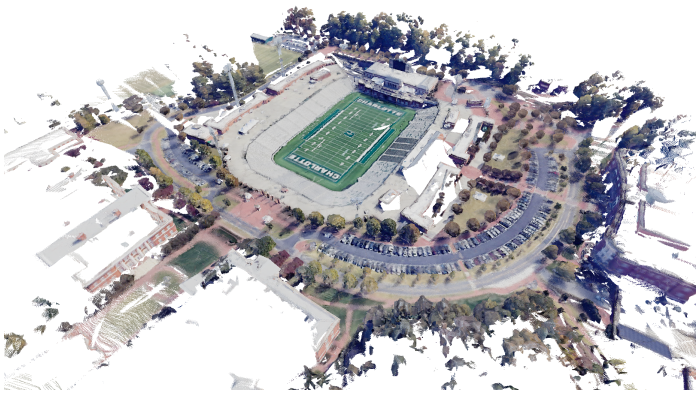
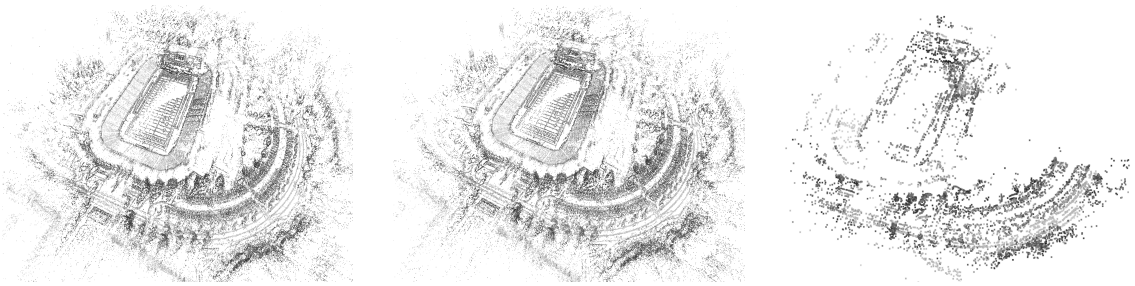


Figure 2.11: The ground truth point cloud of the scene generated by applying the ground truth odometry to the point cloud of each RGB-D frame.



(**a**) (**b**) (**c**)

Figure 2.12: Point clouds generated by (**a**) DSO , (**b**) SDSO, and (**c**) DSOL. DSO and SDSO generated much more point clouds than DSOL. The color of the points is represented by the grayscale color of the scene point.

Quantitative Analysis

Table 2.1 details the density and accuracy characteristics of mapping results obtained from different algorithms. The ICP algorithm was used to align estimate maps with the ground truth point cloud. Criteria for alignment convergence and correspondence calculation included the following: (1) a search radius of 0.5 m, (2) algorithm termination criteria which are triggered when either the RMSE of corresponding points changes by less than 0.00001 or the maximum number of iterations exceeds 1500. Alignment results allow for statistics to be tabulated on the point cloud accuracy for each algorithm. DSO, as a monocular SfM algorithm, is unable to estimate the scene scale accurately. The scale factor was estimated from the ICP algorithm for DSO and the DSO mapping result (point cloud) was scaled by the estimated factor for accuracy evaluation. For SDSO and DSOL, the scale factor was not estimated since the scale of the scene can be derived from the stereo data. In our experiments, the estimated map scale of DSO was 35.98. Table 2.1 captures key statistics of the alignment process for the three algorithms evaluated. Each row of this table is explained below:

- points: Total points in the reconstructed point cloud.
- correspondences: Total amount of correspondences.

- mean: Mean of the distance between all corresponding points.
- std: Standard deviation of the distance between all corresponding points.

Table 2.1: Quantitative evaluation of the point clouds generated by three mapping algorithms.

	DSO	**SDSO**	**DSOL**
points	204345	212179	6662
correspondences	172862	183460	2799
mean (m)	0.110	0.110	0.177
std (m)	0.110	0.111	0.145

Notably, DSO and SDSO maps, similar to each other in the total amount of points, encompass ∼30 times more points than the DSOL map. This aligns with the observations in Figure 2.12. The difference in the correspondence sets is more pronounced, with DSO and SDSO revealing ∼65 times more correspondences than DSOL. DSO, after scaling, follows very closely to SDSO in terms of mapping accuracy performance, while both of them exhibit a 60.9% lower mean error than DSOL and a 31.8% smaller standard deviation. DSO and SDSO prove more accurate and consistent in their mapping results, while DSOL lags in terms of both precision and reliability.

Figure 2.13 depicts the distribution of the distance between corresponding map locations for the three algorithms. DSO and SDSO exhibit similar distributions and most 3D measurements lie within 0.15 m to their corresponding location in the ground truth model. In contrast, DSOL has significantly fewer points within the 0.15 m distance range and a nearly constant number of points having similar errors for greater distances. This supports the mapping accuracy results shown in Table 2.1.

Figure 2.14 indicates the capability of each algorithm to estimate large depths from a given viewpoint (Figure 2.14a) and expected error for a depth estimate for each depth (Figure 2.14b) where depths have been binned to 5 m intervals for tabulation.

Figure 2.14a shows the distribution of depth values for the keyframes of the trajectory which are responsible for generating depth values. Figure 2.14a indicates that a majority of depth estimates range from 20 m to 60 m. One can also see that DSO is capable of generating estimates at larger depths than the two other algorithms (see ranges 100–130 m). DSOL tends to reconstruct points within 60 m and shows a slightly bimodal behavior with a high population of measurements in the 60–100 m range which may be an artifact due to the experimental context. Figure 2.14b portrays the expected depth error in each keyframe. Figure 2.14a also lacks any presence of short ranges. This can be attributed to flying at low-altitude where most data is further than 10 m away.

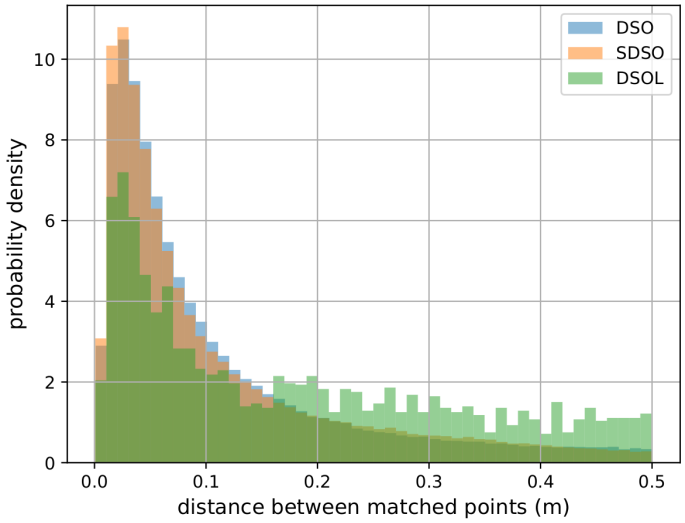
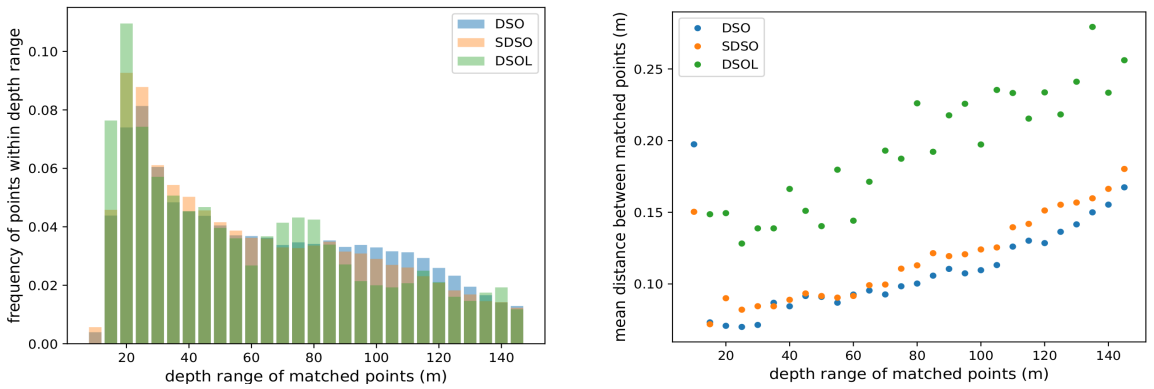


Figure 2.13: Quantitative analysis of the distribution of the closest point distances for correspondences from the mapping results to the ground truth point cloud.



(**a**) (**b**)

Figure 2.14: (**a**) Distribution of point depth in the reconstructed maps and (**b**) the average distance between the matched points at different depth ranges.

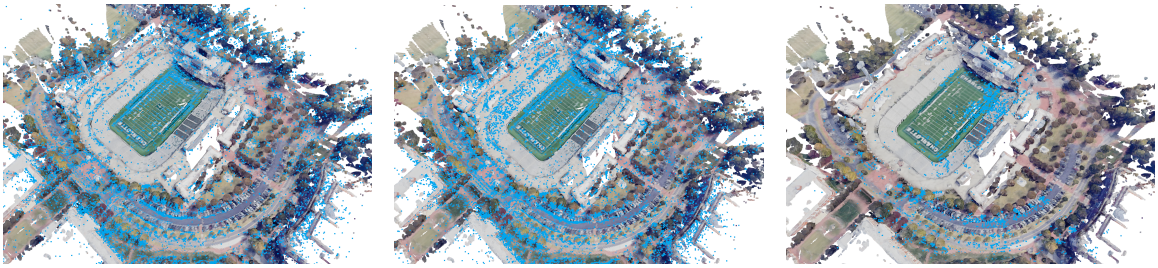
Figure 2.14b, shows the expected depth error for estimate depths. Inspection of the results for distances of 20–40 m, the reconstruction error of DSOL is approximately 0.15 m per point while DSO and SDSO are close to each other having an error of approximately 0.085 m. DSO outperforms SDSO across most depth ranges with slightly smaller distance measurements. Additionally, the error distributions exhibit a quadratic growth pattern as predicted by theoretical models as described in Figure 2.2. High error is noted at short ranges of less than 25 m. This can be attributed to a lack of sufficient supporting image data due to the high velocity of the UAV. Surfaces close to the vehicle move quickly through the field of view and exhibit more motion artifacts leading to higher depth estimation error.

Figure 2.14 indicates that DSO exhibits lower error values across all ranges yet has fewer points. This can be attributed to a strong filter on the acceptable point depth covariance for map points within the algorithm. SDSO and DSOL exhibit lower accuracy compared to those produced by DSO.

Qualitative Analysis

Figure 2.15 shows reconstructed maps from DSO, SDSO, and DSOL. A qualitative examination of these results unveils notable distinctions in their alignment with the

ground truth. DSO and SDSO, with their significantly higher point densities appear to exhibit good accuracy as evidenced by the details of the road network that have been captured and include intricate and well-aligned geometries, e.g., road curbs. The enhanced point density, particularly evident in the football stadium region, allows for a more detailed reconstruction and appears to provide better alignment results relative to ground truth here. Conversely, DSOL, characterized by a sparser point cloud provides a reduced level of detail, particularly in complex structures like the football stadium. Although DSOL shows good alignment for roads, the sparsity of the estimate limits the map details.



(**a**) (**b**) (**c**)

Figure 2.15: Reconstructed point clouds (blue) overlaid with the ground truth point cloud (actual color): (**a**) DSO, (**b**) SDSO, and (**c**) DSOL. DSO point cloud has been scaled by the factor estimated by ICP.

2.1.4.2 Computational Cost Evaluation

Computation cost for the considered algorithms considers the resources required by two critical mapping algorithm functions cost: (1) keyframe creation time and (2) frame tracking time. These metrics serve as crucial benchmarks in assessing the algorithms' ability to swiftly and accurately generate keyframes, as well as tracking real-time camera pose changes during the mapping process. Through this examination, we seek to offer valuable insights that contribute to the informed selection and deployment of mapping solutions for low-altitude UAV flights, particularly for high-speed applications.

Keyframe Creation Time

Figure 2.16 illustrates the keyframe creation time for DSO, SDSO, and DSOL. It can be seen that SDSO requires the most time to create a keyframe, averaging ∼220.73 ms per keyframe, as reported in Table 2.2. DSO incurs lower computational cost for keyframe creation since the stereo disparity map estimation algorithm is not required resulting in keyframe times averaging around ∼200.28 ms per keyframe. DSOL requires ∼7.39 ms per keyframe which is approximately 30 times faster than competing approaches. This can be attributed to the simplified keyframe creation process facilitated as a combination of a simplified disparity computation algorithm and parallel processing. The columns in Table 2.2 delineate the statistical distribution of keyframe creation times, including the minimum, maximum, and mean values, with the "std" column denoting the standard deviation. In summary, SDSO necessitates 10.21% more keyframe creation time than DSO and 2886.87% more than DSOL, while DSO requires 2610.15% more time than DSOL.

Table 2.2: Statistics of the keyframe creation results of three mapping algorithms.

	Total kfs	**min (ms)**	**max (ms)**	**mean (ms)**	**std (ms)**
DSO	330	151.83	274.44	200.28	19.63
SDSO	359	172.19	300.79	220.73	23.08
DSOL	61	1.88	17.99	7.39	2.66

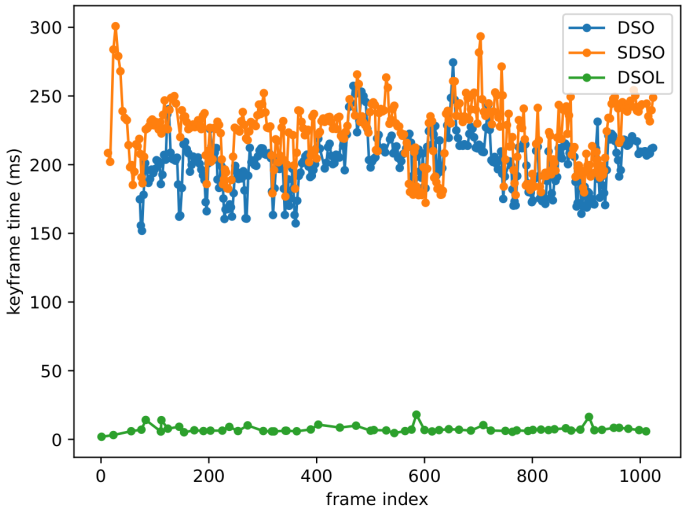
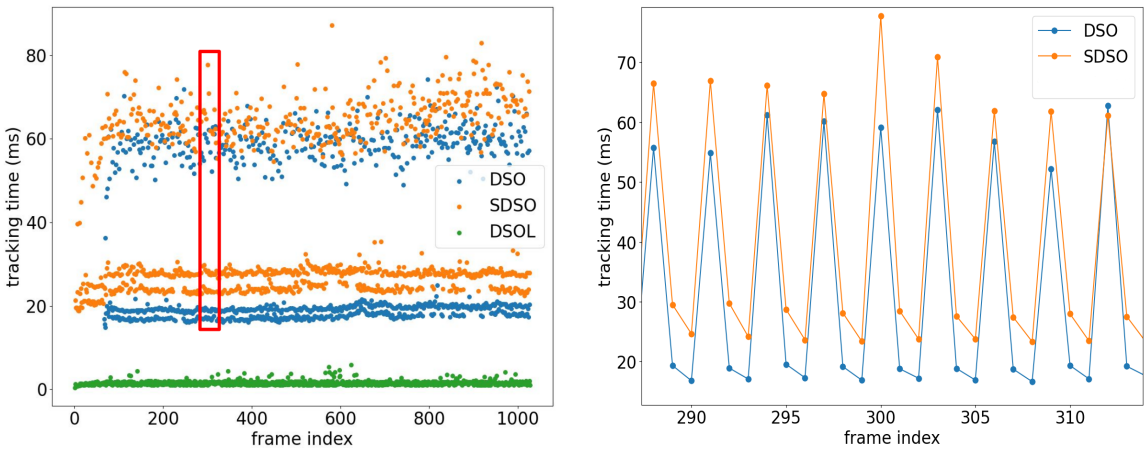


Figure 2.16: The points plotted along the curves represent the keyframe creation time at each frame.

Table 2.2 contains data that provides quantitative measures for the aggregate number of keyframes generated by the three algorithms ("total kfs" column). DSOL generates approximately ∼80% fewer keyframes compared to its counterparts which can be attributed to slightly more restrictive requirements for keyframe creation. Noteworthy is the observation that DSO creates 29 fewer keyframes than SDSO. This discrepancy is attributed to a delayed initialization of the DSO system, commencing at the 50th frame in our experiments, in contrast to the immediate initialization of SDSO and DSOL. Such delay is also illustrated in Figure 2.16 as the DSO curve starts later than SDSO and DSOL. The DSO initialization process relies on assigning random depth values to candidate points and predicting the initial camera movement pattern, demanding precise assumptions about initial depth values and camera motion. In contrast, mapping systems employing stereo cameras, such as SDSO and DSOL, leverage stereo matching for enhanced depth initialization, leading to increased accuracy. Divergence in keyframe quantities among the algorithms also mirrors the disparities in point cloud density depicted in Figure 2.12, given that these points are derived from the keyframes.

Frame Tracking Time

Figure 2.17 depicts the frame tracking time across various algorithms, employing scatter points for visualization. Results show the very high performance achieved by DSOL which requires very little computation for each tracked frame. In the case of DSO and SDSO, the tracking time is stratified into two distinct regions. The upper region, requiring approximately ∼60 ms for tracking, corresponds to keyframes, while the lower region, with an average tracking time of ∼20 ms per frame, pertains to non-keyframes. This 3× difference in tracking time arises from the creation of a new keyframe, where existing point tracks must be terminated and a collection of new point tracks must be initialized incurring significant computational cost to transfer the tracking information. Subsequent frames are then exclusively tracked to this keyframe, employing traditional two-frame direct image alignment methods. This stratification in tracking time offers insights into the computational demands associated with keyframe and non-keyframe tracking, highlighting the intricacies involved in SfM methods that must maintain accurate and efficient tracking across consecutive frames.



(**a**) (**b**)

Figure 2.17: Frame tracking time of different algorithms. (**a**) shows the tracking time for all frames including keyframes and non-keyframes. The data are plotted as scatter points for a clear visualization. (**b**) shows the DSO and SDSO tracking time for frames 290∼310 which corresponds to the region highlighted by the red box in (**a**).

Figure 2.17 also shows two apparent bands in the lower region for results of DSO and SDSO. The higher band characterizes the tracking time for frames immediately succeeding keyframes, while the lower band denotes the tracking time for other frames. A repeated pattern exists where ∼5 ms of addition time is required to process frames following keyframes. Figure 2.17b zooms into a subsection of the data associated with frame indices 290–310. Close examination of this phenomenon indicates that newly formed tracks require more time as the points of the initial keyframe have to be sorted into reliable and unreliable tracks thereby necessitating slightly more computation.

2.1.5 Conclusions

This paper presents a study on low-altitude and high-speed drone applications. An examination of various sensors underscored their strengths and challenges, guiding the selection of suitable devices for specific operational scenarios. The experiments centered on evaluating three prominent mapping algorithms-DSO, SDSO, and DSOL-in a simulated environment, providing valuable insights into the performance of these mapping algorithms. Each algorithm exhibits unique strengths and trade-offs, cater-

ing to specific requirements in UAV-based mapping scenarios. DSO, operating as a monocular mapping algorithm, demonstrates versatility in capturing scenes with a single camera, albeit with limitations in scale estimation. SDSO, incorporating stereo depth perception, excels in accuracy and spatial fidelity, as evidenced by its superior point cloud density and detailed reconstructions, particularly in complex structures like the football stadium. On the other hand, DSOL, designed for efficiency, streamlines the mapping process, offering reliable reconstructions with reduced computational demands. The findings suggest that, in cases where UAVs have limited computing resources, DSOL emerges as the optimal choice. For systems equipped with payload capacity and moderate compute resources, SDSO proves to be the most suitable option. When dealing with a single camera, DSO is the preferred choice for applications demanding dense mapping results.

Future work may involve refining these algorithms for optimized performance in diverse environments, ultimately contributing to advancements in UAV-based mapping for low-altitude and high-speed drone applications. This study contributes to the ongoing discourse on mapping algorithms, providing valuable insights for researchers and practitioners navigating the dynamic landscape of UAV applications in remote sensing and environmental monitoring.

2.2 Photometric Correction for Infrared Sensors

2.2.1 Introduction

Infrared (IR) imaging sensors consist of a grid of radiation-sensitive optoelectronic components that are sensitive to radiated energy having wavelengths from the IR frequency band which spans wavelengths $\lambda \in [0.7\mu m, 1000\mu m]$. The IR frequency band is commonly divided into the NIR (Near IR, $\lambda \in [0.7\mu m, 2.5\mu m]$), MWIR (Mid-Wavelength IR, $\lambda \in [2.5\mu m, 5\mu m]$), LWIR (Long-Wavelength IR, $\lambda \in [8\mu m, 15\mu m]$), and the FIR (Far IR, $\lambda \in [15\mu m, 1000\mu m]$). Thermal infrared (TIR) cameras are a special type of infrared camera which use microbolometer sensors to detect radiation

in the MWIR and LWIR frequency ranges.

Infrared sensors have essential applications in a wide variety of sensing contexts and have recently seen much interest as a sensor to facilitate autonomous ground and aerial vehicle navigation. Heightened interest is largely due to the IR sensor's capability to sense accurate scene image data under low light conditions. This is particularly useful in contexts where illumination is unavailable, e.g., navigating at night and when the visible light sources, surface appearance textures, and surface reflections introduce difficulties to visible light algorithms, e.g., camouflaged targets [55, 56, 57, 58].

Due to the modality difference, the existing visual SLAM frameworks do not directly translate to the thermal domain. Thermal infrared cameras have several characteristics that can cause photometric inconsistency for vision algorithms. Such inconsistency invalidates the "viewpoint invariant" assumption of scene points enforced in most low-level computer vision tasks. First, the rapid automatic gain change (AGC) can incur that the scene point with constant thermal radiant energy appears with different image intensities in consecutive frames [59]. Secondly, microbolometer sensors typically have long response time ($8 \sim 15$ ms) that lead to "ghosting" of sensed values where the value of a pixel in a past frame persists in the current frame creating a spatial-temporal blur of moving objects in sensed IR images.

Similar to related work for RGB cameras [60], photometric correction for thermal infrared sensors promises to improve the accuracy of measured pixels by estimating the underlying scene irradiance responsible for generating a pixel value. Accurate estimates of the scene irradiance promise to improve the output quality of many existing computer vision algorithms [61, 62] including SLAM.

This chapter proposes a photometric correction model and a SLAM system for thermal infrared sensors. The contributions include novel theoretical and experimental results including:

- A novel photometric correction model for thermal infrared sensors is proposed.
- A SLAM system for thermal infrared sensors using this model is developed.
- Experimental work showing the proposed photometric correction model can improve algorithm performance for thermal infrared SLAM problems.

The impact of the proposed photometric correction is important to several sensor types, sensing conditions, and sensing contexts which include: (1) uncooled TIR sensors, (2) TIR sensors that move or observe moving scenes, (3) IR sensors that operate in high-temperature environments and (4) TIR sensors that operate at high frame rates. Each of these circumstances can lead to photometric inconsistency both spatially and temporally. The proposed TIR photometric correction approach seeks to compensate for these effects.

2.2.2 Related Work

2.2.2.1 Infrared Imaging with Microbolometers

An infrared camera is a device that converts infrared radiation into a visual image that depicts temperature variations across an object or scene. Infrared radiation is a characteristic of all objects that have a temperature higher than absolute zero (zero Kelvin or -273 Celsius) [63]. Thermal energy radiated by scene objects is focused onto the sensor image plane where a grid of microbolometer sensors is placed to convert the optical energy focused onto each grid element into a pixel voltage indicative of the object temperature.

The characteristic response of microbolometers is known in the literature [64, 65]. The physical response of each pixel element is governed by a heating and cooling mechanism as the camera shutter is opened and closed. Similar to RGB optical sensors, microbolometer sensors integrate incident radiation into stored charge when the camera shutter is open and dissipate the stored charge when the camera shutter is closed. This process generates a response analogous to an RC circuit driven by a

square wave excitation where the square wave period is determined by the exposure time and its amplitude is determined by the radiated energy of the scene onto the pixel sensor. The rate of integration and dissipation is driven by a sensor-specific time constant, τ.

One shortcoming of microbolometer sensors is their response time. RGB pixel sensors based on CMOS technology have been shown to have time constants $\tau \in [1\mu s, 500\mu s]$ with the median sensor performance across RGB sensors $\tau \approx 10\mu s$ at room temperature [66]. Typical response times for microbolometer sensors are $\tau \in [8ms, 15ms]$ which is more than two orders of magnitude larger. Long response times lead to "ghosting" of sensed values where the value of a pixel in a past frame persists in the current frame creating a spatiotemporal blur of moving objects in sensed IR images.

2.2.2.2 Photometric Correction

Photometric correction for TIR cameras is a new topic in the computer vision field. Many works focus on performing sensor-specific photometric calibration for specific tasks [67, 68]. In [59], an affine sensor response model is proposed to provide a general photometric correction for TIR cameras. A similar affine model is also applied by [69]. Other than these efforts, we have been unable to find other references within the computer vision literature detailing similar approaches.

In contrast, photometric correction has been a well-researched area for visual spectral (RGB) cameras. Early work focuses on estimating the camera response function or vignetting function from only one image [70, 71]. Other works using multiple images require a large number of pixel correspondences to be easily acquired by aligning overlapping image pairs [72, 73]. All of them can only estimate for either camera response or vignetting function, requiring knowledge of the other. To address this problem, [74, 75, 76] propose linear or non-linear optimization frameworks to jointly solve for response and vignetting functions, which have laid the foundation for more

recent work. In [60], a framework without imposing a parametric model is proposed to calibrate a non-parametric response function and vignetting map. In [77], researchers apply the nonlinear estimation formulation in [74] to arbitrary video sequences using gain robust feature tracking, recovering response function, vignetting, exposure times, and radiances of the tracked scene points.

2.2.2.3 Thermal Infrared SLAM

The robustness of motion estimation under low-illumination conditions highlights the thermal infrared camera in the literature. Thermal SLAM methods can be mainly divided into two categories: sensor-fusion and non-fusion. Fusion works combine thermal information with other sensors, such as LiDAR [69, 78, 79], radar [80], visible camera [81, 82, 83, 84], and IMU (inertial measurement unit) [85, 86, 87, 88]. Shin and Kim [69] propose a direct localization and mapping method tracking the sparse depth provided by LiDAR on the raw thermal image. Chen et al. [78] develop a SLAM framework that uses both the raw data and detected features to improve the system's accuracy and robustness. Sehwan et al. An enhanced point cloud generation method in proposed in [79] for indoor odometry under low-visibility conditions by fusing a LiDAR sensor with a stereo thermal infrared camera. Doer and Trommer [80] propose a Kalman-filter-based system that fuses 3D radar ego velocity with monocular thermal inertial odometry. RGB-thermal fusion can make full use of the complementary information of RGB and thermal images, making it an ideal solution for better and more robust image analysis and application tasks under complex illumination [89]. Early work in [81] shows a handheld setup consisting of an RGB-D and a thermal camera for thermographic mapping of objects. However, the underlying odometry estimation relies purely on RGBD data. Recent work [82, 83, 84] detects features in both RGB and thermal images and combines them together for simultaneous location and mapping. Inertial data is also popularly used along with thermal data in the sensor-fusion method [85, 86, 87, 88]. Inertial data in these methods can

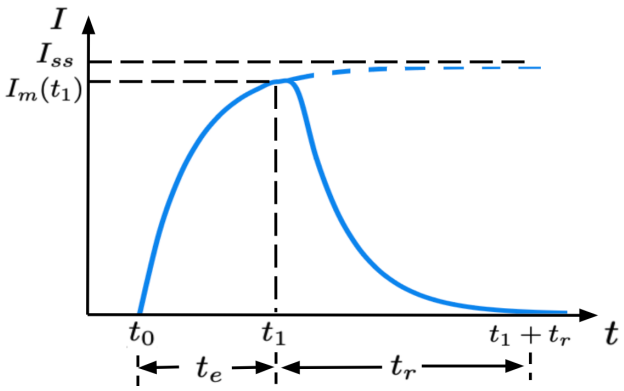
improve the tracking performance when the observation in thermal images is noisy. Non-fusion method focuses on leveraging thermal infrared sensors alone to perform SLAM [90, 91]. However, one of the challenges is that photometric consistency cannot be guaranteed in consecutive frames due to rapid automatic gain change (AGC) [59]. When thermal-infrared cameras capture images of hot objects that move in and out of their field of view, it causes large regions of pixels to become over-saturated. To prevent this, the camera automatically adjusts its gain to darken the image, resulting in significant changes in intensity from one frame to the next. Such a problem also occurs in the fusion methods. In this chapter, we propose a photometric correction model to compensate for the AGC that can be used in any thermal infrared SLAM system to improve performance.

2.2.3 Methodology

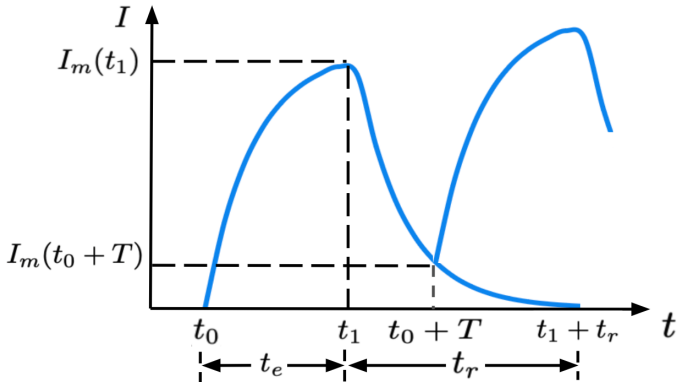
In our previous work published in [92], we proposed a photometric correction model appropriate for microbolometer sensors typically integrated into infrared cameras. A new theoretical model for the pixel response is proposed and the parameters of this model are characterized. The photometric correction model was integrated into a state-of-the-art SfM (Structure-from-Motion) algorithm, DSO (Direct Sparse Odometry) [13], where it was shown to improve upon the structure and camera motion estimates. Prior literature has made clear that photometric correction is an important component in improving the performance of SfM for RGB sensors and the results of this article indicate that appropriate models for infrared photometric correction also improve estimates in the infrared frequency regime. We hypothesize that the impact of the proposed infrared photometric correction further generalizes to potentially improve other computer vision algorithms when applied to IR image sensors, particularly those with the "view invariance" or Lambertian assumption for the radiance of scene points over short motion distances.

Photometric correction is necessary for uncooled infrared image sensors because the

microbolometers used by these sensors have a response that is entirely different from photon detector imagers such as RGB cameras. Specifically, uncooled microbolometer devices require a certain portion of each frame time to integrate signals. Hence for high-speed temperature measurement with short frame time, pixels in a microbolometer usually do not have enough time to reach the temperature of the scene to be measured (a steady temperature state) before the pixels receive new radiance from objects in the next frame. Moreover, microbolometers do not have a mechanism to reset the integrated signal from the previous frame, and therefore signals captured in previous frames will have a residual impact on the microbolometer pixel reading in the current frame. Both of these factors result in an "inaccuracy" of sorts in pixel values in the images generated by infrared sensors as they are not determined solely by the "current scene" and this results in unreliable reconstruction results via structure from motion.



(**a**) Microbolometers heating and cooling process



(**b**) Microbolometers behavior for two consecutive frames

Figure 2.18: An illustration of microbolometer pixels' behavior during the heating and cooling process. (a) Pixels do not have enough time to reach the temperature of the scene being measured. (b) In two (or more) consecutive frames the later frame(s) has a memory of the energy residual in the previous frame(s).

The proposed model for photometric correction of microbolometer measurements consists of two parts: (1) a heating model which characterizes the IR pixel response during a frame exposure and (2) a cooling model which characterizes the IR pixel response when the sensor is not exposed. This section describes these models using continuous differential equations and combines these models into a comprehensive photometric correction model for IR pixels measured at arbitrary framerates. We discuss camera models that allow these corrections to be position invariant and, under these circumstances, algorithms can quickly apply photometric correction across all image pixels using lookup tables to improve their performance. We then integrate this model into the DSO SfM solution and detail how the SfM problem is modified

by the integration of this new photometric correction.

2.2.3.1 Model for Pixel Heating

Heating the IR pixel is modeled as a differential equation with a unit step forcing function, $\mu(t)$, where the amplitude of the step is proportional to the scene irradiance. We then seek to calculate the steady pixel value that reflects the unknown irradiance of the scene point which corresponds to the asymptotic of the step response. The rate of the convergence of the pixel value to the steady state response is determined by the *heating time constant*, τ_h. The process of pixel heating up can be depicted in Fig. 2.18.

The model denotes the initial measurement time as $t = t_0$ and uses the "black body" assumption to set the initial value of the IR pixel, i.e., $I_m(t_0) = 0$.

A first-order differential equation model is provided in Eq. 2.7 to characterize heating an IR pixel.

$$\begin{gathered} \tau_h \frac{dI_m(t)}{dt} + I_m(t) = I_{ss} \\ I_m(t_0) = 0 \end{gathered} \tag{2.7}$$

where $I_m(t)$ is the pixel intensity value measured by the camera sensor at time t, and I_{ss} is the steady state pixel intensity value if the microbolometers were given sufficient heating time.

Let $t_e = t_1 - t_0$ denote the exposure time, meaning that the shutter is open from the beginning time t_0 to time t_1. Solving the first-order differential equation at $t = t_1$ gives

$$I_{ss} = \frac{I_m(t_0 + t_e)}{(1 - e^{-\frac{t_e}{\tau_h}})} \tag{2.8}$$

By modeling the heating process of a microbolometer, we find the value of the forced response at a steady state due to the excitation of the pixel from the associated

portion of the 3D scene.

2.2.3.2 Model for Pixel Cooling

Cooling the IR pixel is modeled as the natural response of the same differential equation after the heating forcing function, $\mu(t)$, has been set to zero. We then seek to calculate the measured pixel value at the time $t = t_0 + T$ where T is the time of each frame. This pixel value will then contribute as a non-zero initial condition for the subsequent image at time $t_0 + T$. The decay rate of the pixel value is determined by the *cooling time constant*, τ_c. The process of pixel cooling is depicted in Fig. 2.18.

The model denotes the excitation of the pixel at the time that the forcing function is removed as $t = t_1$ when the shutter is closed and uses the value of the measured pixel at $t = t_1$ as the initial value of the IR pixel, i.e., $I_m(t_1) = I_0$.

Similarly, another first-order approximation is used to describe the cooling process.

$$\begin{aligned} \tau_c \frac{dI_m(t)}{dt} + I_m(t) &= 0 \\ I_m(t_1) &= I_0 \end{aligned} \tag{2.9}$$

Let $t_r = t_0 + T - t_1$ denote the sensor readout time when the shutter is closed. Solving this first-order differential equation at time $t = t_0 + T$ gives

$$I_m(t_0 + T) = I_0 e^{-\frac{t_r}{\tau_c}} \tag{2.10}$$

By modeling the cooling process of a microbolometer, we find the pixel value at the end of each frame period, which is also the initial measurement for the next frame.

2.2.3.3 Complete Video Sensing Model

We consider sensors that record images at a framerate of f_s or equivalently having a temporal sample period $T = \frac{1}{f_s}$. The time interval between each frame, T, is further subdivided into a measurement or exposure time during which time the shutter is open, t_e, and a readout time during which the shutter is closed, t_r. During the

exposure time period, the microbolometer is heated. During the period that the sensor reads out the pixel values, the microbolometer is cooling. Fig. 2.18 shows the pixel value convergence when the microbolometer is heating and the heat residual from the previous frames left on the new frame when the microbolometer is cooling. We seek to calculate the steady state pixel value excited by a scene point with the prior heat residual removed. Assuming that the effect from previous frames is dominated by the most recent prior frame when there are no drastic temperature gradients in the scenes, the earlier frames are ignored in the model. The complete video sensing model in Eq. 2.11 merges these two models into a comprehensive model for the pixel response, $I'_i(t_e, t_r)$ at frame i while recording a time-sequence of images.

$$I'_i(t_e, t_r) = \frac{I_i - I_{i-1}(e^{-\frac{t_{r,i-1}}{\tau_c}})}{(1 - e^{-\frac{t_{e,i}}{\tau_h}})} \tag{2.11}$$

where I_{i-1} and I_i are two continuous frame, $t_{r,i-1}$ is the readout time in the previous frame $i-1$, and $t_{e,i}$ is the exposure time of the current frame i.

By combining the heating model and cooling model, the irradiance, or the temperature of the measured scene point, can be more accurately reflected by the stable pixel value I'_i.

In the remainder of this chapter, I_i will always refer to the photometrically corrected image I'_i, except where otherwise stated.

2.2.3.4 IR Sensor-Based Structure From Motion (SfM)

In DSO the authors develop advanced photometric correction models for both camera lens and RGB pixel sensing compensation. This is coupled with camera calibration data to perform highly accurate SfM at real-time rates with impressive 3D structure reconstruction results. To leverage such a system and apply it to infrared sensors, the brightness transfer model used for RGB sensors in DSO is replaced by our infrared sensor photometric correction model with the following modifications in the

SfM algorithm.

- Added a time history (previous sensed values) to tracked pixels.
- Modified the optimization approach to use the derivatives and Hessian of our photometric correction model.

To reconstruct a 3D scene from infrared images using structure from motion, a map is computed to associate two pixels in different frames that both correspond to the same 3D scene point. The photometric error between them is defined in a similar way as [13]. For a point, $\mathbf{p}$ in reference frame I_i, observed in target frame I_j, the photometric error, given by Eq. 2.12, is formulated as the weighted Sum of Squared Differences (SSD) over a small neighborhood of pixels.

$$\begin{aligned} E_{\mathbf{p}j} &:= \sum_{\mathbf{p}\in\mathcal{N}_{\mathbf{p}}} w_{\mathbf{p}} |0 I_j\left[\mathbf{p}'\right] - I_{j,o}[\mathbf{p}'] - \beta(I_i[\mathbf{p}] - I_{i,o}[\mathbf{p}'])|0_\gamma \\ I_{j,o}[\mathbf{p}'] &= e^{-\frac{t_{r,j-1}}{\tau_c}} \cdot I_{j-1}[\mathbf{p}'] \\ I_{i,o}[\mathbf{p}'] &= e^{-\frac{t_{r,i-1}}{\tau_c}} \cdot I_{i-1}[\mathbf{p}'] \\ \beta &= \frac{1 - e^{-\frac{t_{e,j}}{\tau_h}}}{1 - e^{-\frac{t_{e,i}}{\tau_h}}} \end{aligned} \tag{2.12}$$

where $\mathcal{N}_{\mathbf{p}}$ is the set of pixels in the SSD, and $|0 \cdot |0_\gamma$ is the Huber norm. In addition to using robust Huber penalties, a gradient-dependent weighting $w_{\mathbf{p}}$ [13] is applied.

To minimize the photometric error between the corresponding points in two frames, the optimizer then optimizes the heating and cooling time constants τ_c and τ_h, instead of the brightness transfer variables in DSO. The optimization is accomplished using the Gauss-Newton algorithm in a sliding window [35].

2.2.4 Results

In this section, the proposed photometric model for infrared sensors is evaluated on two datasets, the FLIR ADAS Dataset v1.3 [93] and the BU-TIV dataset [94]. Both

datasets contain RGB and thermal images for the same scene. The FLIR dataset contains a video sequence of cameras mounted on a vehicle moving in an area during nighttime while the BU-TIV dataset contains a video sequence of a daytime street scene recorded by stationary cameras. The experimental results are obtained on a 32-core Intel Xeon Silver 4110 CPU.

2.2.4.1 Evaluations on FLIR Dataset

The FLIR ADAS Dataset consists of a video sequence of images taken from an IR camera and an RGB camera mounted to the front of a vehicle. The dataset was acquired with an RGB and IR camera mounted on a vehicle (car) where the IR sensor was a Teledyne FLIR Tau 2 thermal camera and the RGB a Teledyne FLIR Blackfly camera. Both RGB and IR videos were recorded at 30 frames per second (fps) under generally clear conditions during the night. Experiments use a subset of the complete ADAS dataset that corresponds to a video sequence of 600 images where the vehicle drives straight down a road at night. Example images from this video sequence are shown in Fig. 2.19. The results are summarized in two experiments:

- Evaluations on the reconstruction quality of the road show that our photometric correction enables DSO to track more points on IR data and improve the accuracy of reconstruction.

- Evaluations on the camera motion show that with the proposed correction model, the trajectory is more stable and less deviated in terms of being at a certain distance away from the RGB camera trajectory.

(**a**) (**b**)

Figure 2.19: An image pair sample from the FLIR ADAS dataset: IR image (left) and RGB image (right).

Evaluation Metrics

Our photometric correction algorithm for IR pixel value correction was integrated into the code for the DSO algorithm [13] as a representation of a state-of-the-art SfM algorithm for RGB image sensors. Experiments are performed using RGB and IR image data as input to the DSO algorithm. We consider 3 outputs: (1) the SfM estimates using the input RGB images, referred to as "*RGB*", (2) the SfM estimates using the input IR images without the proposed correction, referred to as "*IR*", and (3) the SfM estimates using the IR input images with the proposed correction, referred to as "*IR+cor*".

As the only public infrared dataset containing certain data (a series of images taken from different viewpoints) that can be used for solving SfM problems, the FLIR ADAS dataset, however, has some limitations that pose challenges to our experiments: (1) the exposure time information of each frame is not available; (2) the 3D scan of the scene is not provided, and (3) the ground truth of the camera pose is not provided. Missing the accurate exposure time can undermine the advantage of the proposed photometric correction model for IR sensors. The lack of 3D scan and camera pose ground truth makes it difficult to analyze the exact improvements brought by the correction model.

To overcome these drawbacks, based on the appearance of the objects shown in the dataset, for example, the street appears to be flat and the vehicle was driving toward one direction on the same lane, three hypothesizes are made for our experiments:

- **Exposure Time Hypothesis:** The exposure time of each frame is around 10 milliseconds.
- **Planar Road Hypothesis:** The road where the video was recorded can be approximated as a planar surface.
- **Straight-line Trajectory Hypothesis:** The trajectory of the camera, when the vehicle is driving straight on the road, can be approximated as a line.

According to the FLIR Tau2 camera document [95], the exposure time of the Tau2 camera is measured to be about 10 milliseconds. The value is then used as an approximation of the exposure time for each frame in the dataset. The Planar Road Hypothesis allows us to define a plane that fits the structure of the road to serve as a ground truth for the reconstruction quality evaluation. The Straight-line Trajectory Hypothesis provides an opportunity to evaluate the accuracy of the camera pose by looking into the deviation of the trajectory from the line.

Structure Estimate Evaluation

The Planar Road Hypothesis in Section 2.2.4.1 is used for evaluating the structure reconstruction accuracy. The "ground truth" of the road is defined by: (1) segmenting the road area within the point cloud, (2) sampling from each point cloud the **same** amount of points located within the region around the road surface, and (3) merging the points from all three point clouds and fitting a plane to the combined road points. This way a common reference of the road surface is available for evaluation.

The "kfs" and "total pts" rows of Table 2.3 show that the proposed IR photometric correction model allows for more scene points to be tracked for the SfM estimation algorithm. The "kfs" row denotes the total number of keyframes for the image sequence

Table 2.3: Statistics of the road reconstruction performance.

	RGB	IR	**IR+cor**
kfs	219	181	**246**
total pts	55562	42411	**65229**
road pts	1395	1304	**1890**
RMSE	0.0153	0.0137	**0.0119**
SD	0.0088	0.0073	**0.0065**

and the "total" pts row indicates the total number of tracked features for the image sequence. As shown in the table, the SfM algorithm using the IR correction tracks 53.8% more points (total pts) and 35.9% more keyframes (kfs) than IR input images without photometric correction. Similarly, the SfM algorithm using IR correction tracks 17.4% more points and 12.3% more keyframes than RGB input images using the RGB image photometric correction. We also note that similar numbers are found for the point clouds identified as inside the segmented region around the road surface (road points). These results indicate that the IR photometric correction allows more points to be tracked and the resulting SfM solution, therefore, yields a denser 3D point cloud for both the camera motion trajectory (number of keyframes) and the scene structure.

The number of actively tracked features over the 600-frame video sequence is plotted in Fig. 2.20. Actively tracked points are used for both camera motion and scene structure estimation as each new frame in the video is measured. Overall running the "*IR+cor*" enables more points for tracking while both the "*IR*" and the "*RGB*" track fewer points but are comparable to each other. An interesting undulation of the curves shown in the figure occurs at frame index 200. This corresponds to a sequence of images within a large two-way street intersection. The RGB image sensor can track better in this particular context due to rich structural appearance data provided by the white lines and cross-walk textures on the ground which have little thermal contrast. This explains the decrease in tracked points for frames 150-220 from the IR image sensor.

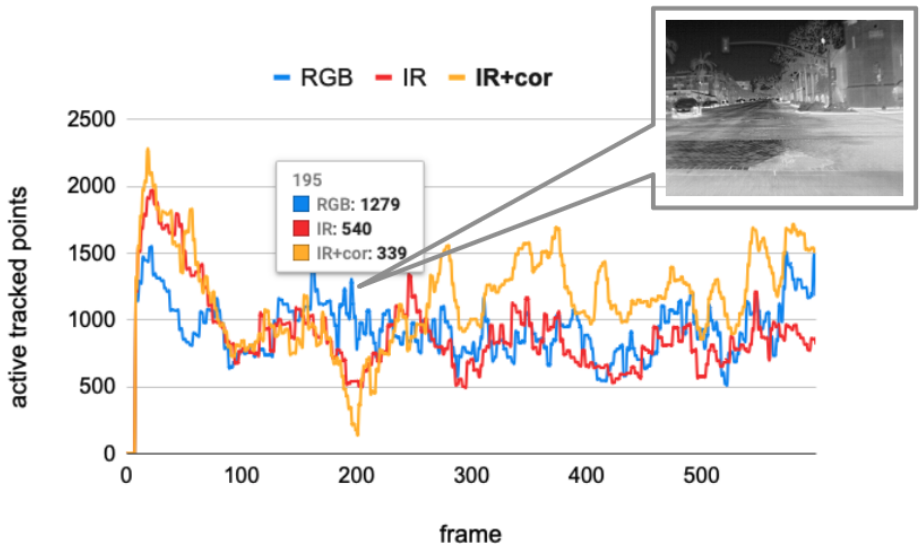


Figure 2.20: The amount of tracked features for each video frame is plotted for "*RGB*" (blue), "*IR*" (red), and "*IR+cor*" (yellow) SfM estimates. The plot shows that the proposed IR photometric correction allows more stable feature detection which results in more tracked features leading to denser SfM estimates.

To evaluate the accuracy of the structure reconstruction identical sections of the estimated reconstruction data in the vicinity of the road surface were segmented from the complete structure estimate. In each case, the segmented surface points were compared against a pre-defined road plane. Evaluation of performance was done by computing the Root-Mean-Square deviation (RMSE) and the standard deviation (SD) of the perpendicular distance between reconstructed 3D points and the road surface. As presented in Table 2.3, "*IR+cor*" improves the reconstruction accuracy of the road by 15.1% over the "*IR*" approach and 28.5% over the "*RGB*" approach. The road points detected in "*IR+cor*" exhibit less noise relative to the road plane model having 10.9% and 26.1% less deviation than the results of the "*IR*" and "*RGB*" methods respectively. These results indicate that the proposed IR photometric correction model reduces the noise in the estimated scene structure.

This conclusion can be further supported by the histogram of the fitting error and error distribution in Fig. 2.21, from which we can also see that DSO on the

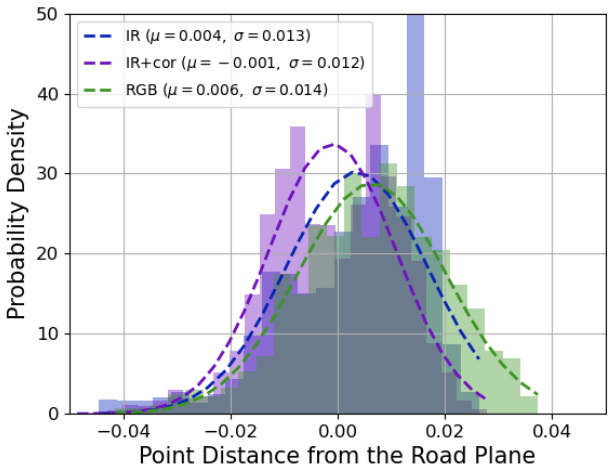


Figure 2.21: Histogram and approximated Gaussian distributions of the fitting error of SfM-estimated 3D points to a common planar surface (the road plane). RGB (green) and IR (blue) show more error variance than the proposed IR photometric correction approach (purple).

IR data with infrared photometric correction achieves the best performance in terms of accuracy (closest to zero mean value) and stability (least standard deviation). Although RGB data can sometimes provide more features for detection and tracking (more tracked points), the reconstruction quality is not as well as the results from infrared data. This can be because the RGB intensity values are very prone to bad illumination conditions such as at night, which was the case when the dataset was recorded.

Motion Estimate Evaluation

Performance analysis for the motion estimates uses a Straight-line Trajectory Hypothesis in Section 2.2.4.1 for the vehicle motion and examines the observed errors of the estimated camera motion for each approach. In the video, the vehicle where the two cameras were mounted was driving on a straight street, as a result of which the trajectory of the cameras can be approximated to be straight as well. Towards this end, a 3D line segment is fit to the positions of the estimated camera trajectories for three approaches: (1) *IR*, (2) *IR+cor*, and (3) *RGB*. Table 2.4 shows the RMSE

(root-mean-square error) and standard deviation (SD) of the perpendicular distances between camera positions and the estimated 3D line model.

Table 2.4: Statistics of the trajectory estimation performance.

	RGB	IR	**IR+cor**
RMSE	0.0128	0.0109	**0.0106**
SD	0.0061	0.0049	**0.0045**

The RMSE error row of Table 2.4 indicates that the camera motion estimates for the infrared data using the proposed photometric correction model exhibit less error relative to the line model by a factor of 2.8% for the IR method and 17.2% for the RGB method. The standard deviation (SD) row of Table 2.4 indicates that the variability in the camera motion for the IR photometric correction is also less than that for the other two approaches. The reduction in noise for both IR estimates relative to the RGB data suggests that IR image data in this low-light context may provide advantages over RGB data for SfM estimation and that the proposed photometric correction further improves the SfM estimation performance in accuracy and stability.

Qualitative Analysis

The point cloud reconstructions from the IR and RGB data are illustrated in Fig. 2.22 respectively. The green box in both Fig. 2.22 and Fig 2.22 includes the building of our interest. The red and blue boxes in the other three figures highlight the area where the reconstruction results differ. Compared to Fig. 2.22, Fig. 2.22 shows sharper edges on the windows in the building (red box) and contains more points to reflect the edge of the top of the building (blue box). This indicates that the photometric correction model we propose can enable SfM algorithms to track more features (points) for reconstruction and the features are less noisy and more stable. As shown in Fig. 2.22, the top of the building (blue box) is completely missing and unrecognizable, and the point cloud of the windows in the building is very noisy (red box). This is because the top area of the building is interfered with by the illumination from the

street light and RGB sensors can easily suffer from such illumination conditions and will fail to detect reliable features. The infrared sensors, however, are more robust in this scenario as they are sensitive to thermal contrast instead of photons. Note that the example image here comes from the same intersection area in the street as the image illustrated in Fig. 2.20, the point clouds here further prove that in this area of the street, more active points are tracked in the RGB data than the IR data due to the fact that rich structure but little thermal contrast are available in this area, as explained previously in Section 2.2.4.1.

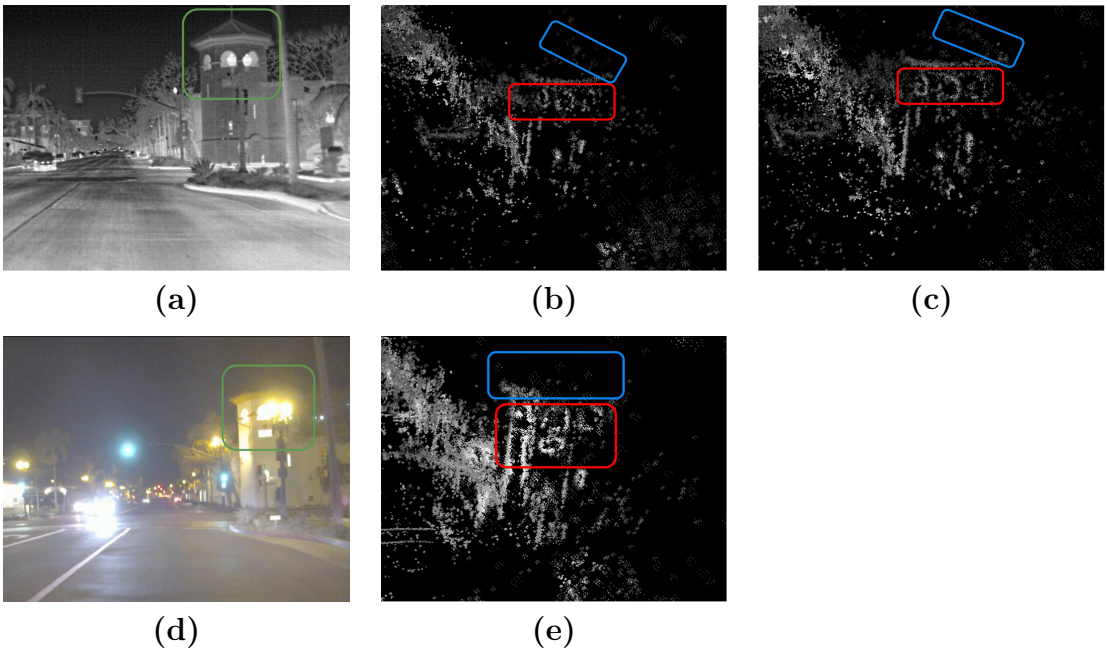


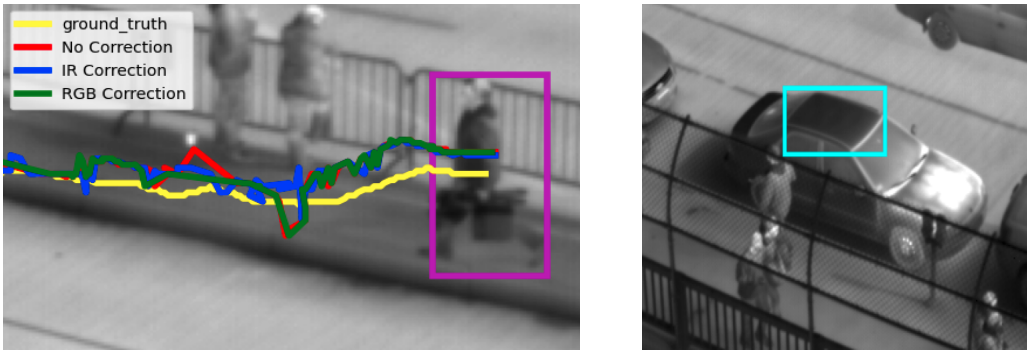
Figure 2.22: **Qualitative illustration.** (a) Infrared image from the FLIR ADAS Dataset. (b) "*IR*" point cloud reconstruction. (c) "*IR+cor*" point cloud reconstruction. (d) RGB image of the same scene from the dataset. (e) "*RGB*" point cloud reconstruction.

2.2.4.2 Evaluations on BU-TIV Dataset

To show that the proposed photometric correction model can also be applied to other IR image-processing contexts, the proposed photometric correction model is applied to the BU-TIV dataset for solving a human being tracking problem. The BU-TIV dataset is designated for the object-tracking problem using infrared data

and consists of video sequences in different scenes that were recorded with FLIR SC8000 cameras. A subset of the dataset of a daytime crowded street view during a marathon competition was used. The results are summarized in two experiments: (1) Experiment 1 tracks pedestrians in IR image sequences. (2) Experiment 2 tracks the observed temperature for a stationary target in the IR video.

Experiment 1 evaluates the proposed photometric correction for pedestrian tracking from an IR image sequence. Results are shown in Fig. 2.23 for three distinct photometric correction approaches: (1) RGB correction (RGB), (2) no correction (IR), and (3) the proposed correction (IR+cor). Table 2.5 shows the quality of each estimate as measured by three distinct metrics: (1) the Discrete Frechet (DF) distance [96], (2) the Dynamic Time Warping (DTW) [97], and (3) a custom metric referred to as the Mean Distance. Mean distance calculates the average distance between the person's estimated and actual positions for all corresponding frames. Table 2.5 indicates that the proposed IR correction improves the performance across all three metrics, where lower scores are better. The last row shows the number of frames where the person is tracked and again the proposed IR correction algorithm outperforms the other cases.



(**a**) (**b**)

Figure 2.23: Excerpts from the IR video in the BU-TIV dataset. (a) shows pedestrian (magenta box) tracking results relative to the ground truth (yellow), for the proposed correction (blue), RGB correction (green), and no correction (red). Table 2.5 provides quantitative performance metrics. (b) shows the roof of the parked car (cyan box) where the temperature is tracked.

Table 2.5: Tracking differences measured by different algorithms between the estimated trajectories and ground truth.

	RGB	IR	**IR+cor**
DF [96]	23.77	39.82	**22.84**
DTW [97]	4444.43	4339.27	**4257.07**
Mean Distance	8.81	8.82	**8.51**
Tracked frames count	496	537	**555**

Experiment 2 tracks the temperature of a parked car roof as shown in Fig. 2.23 and seeks to analyze the stability of the temperature before and after applying the proposed IR photometric correction. Table 2.6 shows that the proposed photometric correction improves the stability of pixel intensity and, by extension, the estimate of the unknown constant temperature of the observed object.

Table 2.6: Standard deviation of observed intensities for the roof of a parked car denoted as a cyan box in Fig. 2.23 over time.

	RGB	IR	**IR+cor**
SD	0.884	0.918	**0.877**

2.2.5 Conclusion

This chapter proposes a photometric correction model appropriate for microbolometer sensors typically integrated into infrared cameras. A new theoretical model for the pixel response is proposed and the parameters of this model are characterized. The photometric correction model was integrated into a state-of-the-art SLAM algorithm where it was shown to improve upon the localization and mapping estimates. Prior literature has made clear that photometric correction is an important component in improving the performance of SLAM for RGB sensors and the results of this article indicate that appropriate models for infrared photometric correction also improve estimates in the infrared frequency regime. We hypothesize that the impact of the proposed infrared photometric correction further generalizes to potentially improve

other computer vision algorithms when applied to IR image sensors, particularly those with the "view invariance" or Lambertian assumption for the radiance of scene points over short motion distances.

REFERENCES

[1] N. Kakavitsas, A. Willis, J. M. Conrad, and A. Wolek, "Comparison of size and performance of small vertical and short takeoff and landing uas," in *2024 IEEE Aerospace Conference*, IEEE, 2024.

[2] R. Szeliski, *Computer vision: Algorithms and applications*. Springer Nature, 2022.

[3] M. Spetsakis and J. Y. Aloimonos, "A multi-frame approach to visual motion perception," *International Journal of Computer Vision*, vol. 6, no. 3, pp. 245–255, 1991.

[4] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustmentâa modern synthesis," in *Vision Algorithms: Theory and Practice: International Workshop on Vision Algorithms Corfu, Greece, September 21–22, 1999 Proceedings*, pp. 298–372, Springer, 2000.

[5] M. I. Lourakis and A. A. Argyros, "Sba: A software package for generic sparse bundle adjustment," *ACM Transactions on Mathematical Software (TOMS)*, vol. 36, no. 1, pp. 1–30, 2009.

[6] D. J. Crandall, A. Owens, N. Snavely, and D. P. Huttenlocher, "Sfm with mrfs: Discrete-continuous optimization for large-scale structure from motion," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 12, pp. 2841–2853, 2012.

[7] B. U. Meinen and D. T. Robinson, "Mapping erosion and deposition in an agricultural landscape: Optimization of uav image acquisition schemes for sfm-mvs," *Remote Sensing of Environment*, vol. 239, p. 111666, 2020.

[8] D. B. Gennery, "A stereo vision system for an autonomous vehicle.," in *IJCAI*, pp. 576–582, 1977.

[9] T. Cao, Z.-Y. Xiang, and J.-L. Liu, "Perception in disparity: An efficient navigation framework for autonomous vehicles with stereo cameras," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 5, pp. 2935–2948, 2015.

[10] A. Geiger, J. Ziegler, and C. Stiller, "Stereoscan: Dense 3d reconstruction in real-time," in *2011 IEEE Intelligent Vehicles Symposium (IV)*, pp. 963–968, IEEE, 2011.

[11] C. H. Esteban and F. Schmitt, "Silhouette and stereo fusion for 3d object modeling," *Computer Vision and Image Understanding*, vol. 96, no. 3, pp. 367–392, 2004.

[12] O. Krutikova, A. Sisojevs, and M. Kovalovs, "Creation of a depth map from stereo images of faces for 3d model reconstruction," *Procedia Computer Science*, vol. 104, pp. 452–459, 2017.

[13] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 3, pp. 611–625, 2017.

[14] R. Wang, M. Schworer, and D. Cremers, "Stereo dso: Large-scale direct sparse visual odometry with stereo cameras," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3903–3911, 2017.

[15] C. Qu, S. S. Shivakumar, I. D. Miller, and C. J. Taylor, "Dsol: A fast direct sparse odometry scheme," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 10587–10594, IEEE, 2022.

[16] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge University Press, 2003.

[17] A. Eltner and G. Sofia, "Structure from motion photogrammetric technique," in *Developments in Earth surface processes*, vol. 23, pp. 1–24, Elsevier, 2020.

[18] S. M. Hasheminasab, T. Zhou, and A. Habib, "Gnss/ins-assisted structure from motion strategies for uav-based imagery over mechanized agricultural fields," *Remote Sensing*, vol. 12, no. 3, p. 351, 2020.

[19] M. Kalacska, J. P. Arroyo-Mora, and O. Lucanus, "Comparing uas lidar and structure-from-motion photogrammetry for peatland mapping and virtual reality (vr) visualization," *Drones*, vol. 5, no. 2, p. 36, 2021.

[20] J. Mooser, S. You, U. Neumann, and Q. Wang, "Applying robust structure from motion to markerless augmented reality," in *2009 Workshop on Applications of Computer Vision (WACV)*, pp. 1–8, IEEE, 2009.

[21] F. Mumuni and A. Mumuni, "Bayesian cue integration of structure from motion and cnn-based monocular depth estimation for autonomous robot navigation," *International Journal of Intelligent Robotics and Applications*, vol. 6, no. 2, pp. 191–206, 2022.

[22] A. Zhanabatyrova, C. S. Leite, and Y. Xiao, "Structure from motion-based mapping for autonomous driving: Practice and experience," *ACM Transactions on Internet of Things*, vol. 5, no. 1, pp. 1–25, 2024.

[23] D. Turner, A. Lucieer, and C. Watson, "An automated technique for generating georectified mosaics from ultra-high resolution unmanned aerial vehicle (uav) imagery, based on structure from motion (sfm) point clouds," *Remote sensing*, vol. 4, no. 5, pp. 1392–1410, 2012.

[24] R. Fujiwara, T. Kikawada, and Y. Akiyama, "Comparison of remote sensing methods for plant heights in agricultural fields using unmanned aerial vehicle-based structure from motion," *Frontiers in Plant Science*, vol. 13, p. 886804, 2022.

[25] G. Caroti, I. Martínez-Espejo Zaragoza, and A. Piemonte, "Accuracy assessment in structure from motion 3d reconstruction from uav-born images: The influence of the data processing methods," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 40, pp. 103–109, 2015.

[26] S. I. Deliry and U. Avdan, "Accuracy of unmanned aerial systems photogrammetry and structure from motion in surveying and mapping: a review," *Journal of the Indian Society of Remote Sensing*, vol. 49, no. 8, pp. 1997–2017, 2021.

[27] P. Lindenberger, P.-E. Sarlin, V. Larsson, and M. Pollefeys, "Pixel-perfect structure-from-motion with featuremetric refinement," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 5987–5997, 2021.

[28] H. Cui, T. Shi, J. Zhang, P. Xu, Y. Meng, and S. Shen, "View-graph construction framework for robust and efficient structure-from-motion," *Pattern Recognition*, vol. 114, p. 107712, 2021.

[29] A. Islam, M. Asikuzzaman, M. O. Khyam, M. Noor-A-Rahim, and M. R. Pickering, "Stereo vision-based 3d positioning and tracking," *IEEE Access*, vol. 8, pp. 138771–138787, 2020.

[30] S. Pillai, S. Ramalingam, and J. J. Leonard, "High-performance and tunable stereo reconstruction," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3188–3195, IEEE, 2016.

[31] N. Kemsaram, A. Das, and G. Dubbelman, "A stereo perception framework for autonomous vehicles," in *2020 IEEE 91st vehicular technology conference (VTC2020-Spring)*, pp. 1–6, IEEE, 2020.

[32] J. Liu, P. Ji, N. Bansal, C. Cai, Q. Yan, X. Huang, and Y. Xu, "Planemvs: 3d plane reconstruction from multi-view stereo," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8665–8675, 2022.

[33] Z. Shang and Z. Shen, "Topology-based uav path planning for multi-view stereo 3d reconstruction of complex structures," *Complex & Intelligent Systems*, vol. 9, no. 1, pp. 909–926, 2023.

[34] P. Irmisch, *Camera-based distance estimation for autonomous vehicles*. PhD thesis, Technische Universität Berlin, 2017.

[35] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual–inertial odometry using nonlinear optimization," *The International Journal of Robotics Research*, vol. 34, no. 3, pp. 314–334, 2015.

[36] S. Baker and I. Matthews, "Lucas-kanade 20 years on: A unifying framework," *International journal of computer vision*, vol. 56, pp. 221–255, 2004.

[37] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2149–2154, 2004.

[38] Robotis and Perception Group – University of Zurich, "Agilicious." https://github.com/uzh-rpg/agilicious. (accessed: Sep. 25, 2023).

[39] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, *Robot Operating System (ROS): The complete reference (volume 1)*, ch. 23, pp. 595–625. New York, NY, USA: Springer, Cham, 2016.

[40] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, "Flightmare: A flexible quadrotor simulator," in *Proceedings of the 4th Conference on Robot Learning*, pp. 1–11, 2020.

[41] K. Mohta, M. Watterson, Y. Mulgaonkar, S. Liu, C. Qu, A. Makineni, K. Saulnier, K. Sun, A. Zhu, J. Delmerico, D. Thakur, K. Karydis, N. Atanasov, G. Loianno, D. Scaramuzza, K. Daniilidis, C. J. Taylor, and V. Kumar, "Fast, autonomous flight in GPS-denied and cluttered environments," *Journal of Field Robotics*, vol. 35, no. 1, pp. 101–120, 2018.

[42] W. Guerra, E. Tal, V. Murali, G. Ryou, and S. Karaman, "Flightgoggles: Photorealistic sensor simulation for perception-driven robotics using photogrammetry and virtual reality," in *Proceedings of the 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 6941–6948, 2019.

[43] C. Beam, J. Zhang, N. Kakavitsas, C. Hague, A. Wolek, and A. Willis, "Cesium tiles for high-realism simulation and comparing slam results in corresponding virtual and real-world environments," 2024.

[44] J. Li, B. Yang, C. Chen, and A. Habib, "Nrli-uav: Non-rigid registration of sequential raw laser scans and images for low-cost uav lidar point cloud quality improvement," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 158, pp. 123–145, 2019.

[45] T.-M. Nguyen, M. Cao, S. Yuan, Y. Lyu, T. H. Nguyen, and L. Xie, "Viral-fusion: A visual-inertial-ranging-lidar sensor fusion approach," *IEEE Transactions on Robotics*, vol. 38, no. 2, pp. 958–977, 2021.

[46] Y.-C. Lin, Y.-T. Cheng, T. Zhou, R. Ravi, S. M. Hasheminasab, J. E. Flatt, C. Troy, and A. Habib, "Evaluation of uav lidar for mapping coastal environments," *Remote Sensing*, vol. 11, no. 24, p. 2893, 2019.

[47] K.-W. Chiang, G.-J. Tsai, Y.-H. Li, and N. El-Sheimy, "Development of lidar-based uav system for environment reconstruction," *IEEE Geoscience and Remote Sensing Letters*, vol. 14, no. 10, pp. 1790–1794, 2017.

[48] "Event cameras comparison." https://inivation.com/wp-content/uploads/2022/10/2022-09-iniVation-devices-Specifications.pdf. Accessed: 2024-03-14.

[49] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics: Results of the 11th International Conference*, pp. 621–635, Springer, 2018.

[50] P. J. Besl and N. D. McKay, "Method for registration of 3-d shapes," in *Sensor fusion IV: control paradigms and data structures*, vol. 1611, pp. 586–606, Spie, 1992.

[51] T. Zodage, "Point cloud registration as a classification problem," Master's thesis, Carnegie Mellon University, Pittsburgh, PA, August 2021.

[52] "Direct sparse odometry official implementation." https://github.com/JakobEngel/dso. Accessed: 2024-01-02.

[53] "Direct sparse odometry lite official implementation." https://github.com/versatran01/dsol. Accessed: 2024-01-02.

[54] "Stereo direct sparse odometry non-official implementation." https://github.com/JiatianWu/stereo-dso. Accessed: 2024-01-02.

[55] R. Grimming, B. McIntosh, A. Mahalanobis, and R. G. Driggers, "Lwir sensor parameters for deep learning object detectors," *OSA Continuum*, vol. 4, no. 2, pp. 529–541, 2021.

[56] N. Pinchon, O. Cassignol, A. Nicolas, F. Bernardin, P. Leduc, J.-P. Tarel, R. Brémond, E. Bercier, and J. Brunet, "All-weather vision for automotive safety: which spectral band?," in *International Forum on Advanced Microsystems for Automotive Applications*, pp. 3–15, Springer, 2018.

[57] A. Rankin, A. Huertas, L. Matthies, M. Bajracharya, C. Assad, S. Brennan, P. Bellutta, and G. W. Sherwin, "Unmanned ground vehicle perception using thermal infrared cameras," in *Unmanned Systems Technology XIII*, vol. 8045, pp. 19–44, Spie, 2011.

[58] S. Kim, W.-J. Song, and S.-H. Kim, "Infrared variation optimized deep convolutional neural network for robust automatic ground target recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 1–8, 2017.

[59] M. P. Das, L. Matthies, and S. Daftry, "Online photometric calibration of automatic gain thermal infrared cameras," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2453–2460, 2021.

[60] J. Engel, V. Usenko, and D. Cremers, "A photometrically calibrated benchmark for monocular visual odometry," *arXiv preprint arXiv:1607.02555*, 2016.

[61] J. K. Suhr, "Kanade-lucas-tomasi (klt) feature tracker," *Computer Vision (EEE6503)*, pp. 9–18, 2009.

[62] S. S. Beauchemin and J. L. Barron, "The computation of optical flow," *ACM computing surveys (CSUR)*, vol. 27, no. 3, pp. 433–466, 1995.

[63] M. Wilson, "Temperature measurement," *Anaesthesia & Intensive Care Medicine*, vol. 22, no. 3, pp. 202–207, 2021.

[64] N. Boudou, A. Durand, S. Tinnes, S. Cortial, A. Gorecki, M. Cueff, and A. Virot, "Ulis bolometer improvements for fast imaging applications," in *Infrared Technology and Applications XLV*, vol. 11002, pp. 366–375, SPIE, 2019.

[65] M. Kohin and N. R. Butler, "Performance limits of uncooled vox microbolometer focal plane arrays," in *Infrared Technology and Applications XXX*, vol. 5406, pp. 447–453, SPIE, 2004.

[66] C. Y.-P. Chao, H. Tu, T. Wu, K.-Y. Chou, S.-F. Yeh, and F.-L. Hsueh, "Cmos image sensor random telegraph noise time constant extraction from correlated to uncorrelated double sampling," *IEEE Journal of the Electron Devices Society*, vol. 5, no. 1, pp. 79–89, 2017.

[67] C. Papachristos, F. Mascarich, and K. Alexis, "Thermal-inertial localization for autonomous navigation of aerial robots through obscurants," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 394–399, IEEE, 2018.

[68] H. Budzier and G. Gerlach, "Calibration of uncooled thermal infrared cameras," *Journal of Sensors and Sensor Systems*, vol. 4, no. 1, pp. 187–197, 2015.

[69] Y.-S. Shin and A. Kim, "Sparse depth enhanced direct thermal-infrared slam beyond the visible spectrum," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2918–2925, 2019.

[70] Y. Zheng, S. Lin, C. Kambhamettu, J. Yu, and S. B. Kang, "Single-image vignetting correction," *IEEE transactions on pattern analysis and machine intelligence*, vol. 31, no. 12, pp. 2243–2256, 2008.

[71] S. Lin and L. Zhang, "Determining the radiometric response function from a single grayscale image," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, pp. 66–73, IEEE, 2005.

[72] S. J. Kim and M. Pollefeys, "Robust radiometric calibration and vignetting correction," *IEEE transactions on pattern analysis and machine intelligence*, vol. 30, no. 4, pp. 562–576, 2008.

[73] M. Brown and D. G. Lowe, "Automatic panoramic image stitching using invariant features," *International journal of computer vision*, vol. 74, pp. 59–73, 2007.

[74] D. B. Goldman, "Vignette and exposure calibration and compensation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 12, pp. 2276–2288, 2010.

[75] A. Litvinov and Y. Y. Schechner, "Radiometric framework for image mosaicking," *JOSA A*, vol. 22, no. 5, pp. 839–848, 2005.

[76] A. Litvinov and Y. Y. Schechner, "Addressing radiometric nonidealities: A unified framework," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, pp. 52–59, IEEE, 2005.

[77] P. Bergmann, R. Wang, and D. Cremers, "Online photometric calibration of auto exposure video for realtime visual odometry and slam," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 627–634, 2017.

[78] W. Chen, Y. Wang, H. Chen, and Y. Liu, "Eil-slam: Depth-enhanced edge-based infrared-lidar slam," *Journal of Field Robotics*, vol. 39, no. 2, pp. 117–130, 2022.

[79] S. Rho, S. M. Park, J. Pyo, M. Lee, M. Jin, and S.-C. Yu, "Lidar-stereo thermal sensor fusion for indoor disaster environment," *IEEE Sensors Journal*, vol. 23, no. 7, pp. 7816–7827, 2023.

[80] C. Doer and G. F. Trommer, "Radar visual inertial odometry and radar thermal inertial odometry: Robust navigation even in challenging visual conditions," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 331–338, IEEE, 2021.

[81] S. Vidas, P. Moghadam, and M. Bosse, "3d thermal mapping of building interiors using an rgb-d and thermal camera," in *2013 IEEE international conference on robotics and automation*, pp. 2311–2318, IEEE, 2013.

[82] L. Chen, L. Sun, T. Yang, L. Fan, K. Huang, and Z. Xuanyuan, "Rgb-t slam: A flexible slam framework by combining appearance and thermal information," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5682–5687, IEEE, 2017.

[83] Y. Ni, Y. Wang, S. Yang, R. Chen, and X. Wang, "Large field of view cooperative infrared and visible spectral sensor for visual odometry," *IEEE Access*, vol. 8, pp. 74237–74249, 2020.

[84] S. Khattak, C. Papachristos, and K. Alexis, "Visual-thermal landmarks and inertial fusion for navigation in degraded visual environments," in *2019 IEEE Aerospace Conference*, pp. 1–9, IEEE, 2019.

[85] J. Jiang, X. Chen, W. Dai, Z. Gao, and Y. Zhang, "Thermal-inertial slam for the environments with challenging illumination," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 8767–8774, 2022.

[86] M. R. U. Saputra, C. X. Lu, P. P. B. de Gusmao, B. Wang, A. Markham, and N. Trigoni, "Graph-based thermal–inertial slam with probabilistic neural networks," *IEEE Transactions on Robotics*, vol. 38, no. 3, pp. 1875–1893, 2021.

[87] Y. Wang, H. Chen, Y. Liu, and S. Zhang, "Edge-based monocular thermal-inertial odometry in visually degraded environments," *IEEE Robotics and Automation Letters*, vol. 8, no. 4, pp. 2078–2085, 2023.

[88] S. Khattak, C. Papachristos, and K. Alexis, "Keyframe-based thermal–inertial odometry," *Journal of Field Robotics*, vol. 37, no. 4, pp. 552–579, 2020.

[89] K. Song, Y. Zhao, L. Huang, Y. Yan, and Q. Meng, "Rgb-t image analysis technology and application: A survey," *Engineering Applications of Artificial Intelligence*, vol. 120, p. 105919, 2023.

[90] S. Vidas and S. Sridharan, "Hand-held monocular slam in thermal-infrared," in *2012 12th International Conference on Control Automation Robotics & Vision (ICARCV)*, pp. 859–864, IEEE, 2012.

[91] T. Mouats, N. Aouf, L. Chermak, and M. A. Richardson, "Thermal stereo odometry for uavs," *IEEE Sensors Journal*, vol. 15, no. 11, pp. 6335–6347, 2015.

[92] J. Zhang, A. R. Willis, and K. Brink, "Photometric correction for infrared sensors," *arXiv preprint arXiv:2304.03930*, 2023.

[93] "TELEDYNE FLIR Dataset." https://flir.box.com/s/suwst0b3k9rko35homhr3rnyytf3102d. Accessed: 2022-05-31.

[94] Z. Wu, N. Fuller, D. Theriault, and M. Betke, "A thermal infrared video benchmark for visual analysis," in *CVPR PBVS Workshop*, pp. 201–208, 2014.

[95] "Time Constant Design of Tau2 and Quark2." https://flir.custhelp.com/app/answers/detail/a$_i$*d*/3171/*related*/1. *Accessed* : $2022 - 11 - 06$.

[96] T. Eiter and H. Mannila, "Computing discrete fréchet distance," 1994.

[97] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series.," in *KDD workshop*, vol. 10, pp. 359–370, Seattle, WA, USA:, 1994.

CHAPTER 3: LOW-BANDWIDTH AND COMPUTE-BOUND SLAM

3.1 Introduction

Many autonomous agent tasks require robust localization and a good representation of the environment, especially when GPS is unavailable. Mobile robots for indoor navigation often operate in structured environments hence indoor navigation focuses more on generating a 2D map. For more complicated tasks, robots need more mobility to operate in more dynamic environments such as slopes, stairs, and tunnels [1, 2, 3]. In these scenarios, robots are usually equipped with LIDAR and high-quality IMU sensors to estimate the poses and the 3D map [4, 5, 6]. However, these sensors are quite expensive and relatively fragile. The emergence of modern consumer RGB-D sensors has had a significant impact on the robotic research fields. They are low-cost, low-power, and low-size alternatives to traditional range sensors such as LiDAR [7]. RGB-D sensors also provide additional depth information which enhances the ability of robots to sense the environment and estimate its structure for navigation. Motivated by RGB-D cameras, a lot of researchers have worked on 3D Visual SLAM, such as monocular SLAM [8, 9], RGB-D SLAM [10, 11], and stereo SLAM [12, 13].

The motivation of our work is to construct a compact representation of maps that can be efficiently shared among robots. Multi-agent systems have demonstrated various applications, such as self-driving cars, warehouse robots, and so on. These applications require collaborative mapping in multi-agent environments where each agent needs to share information about their past and current state. The use of SLAM to generate high-quality 2D or 3D maps is a classical subject and recent work focuses on real-time applications of this algorithm in distributed environments to generate and update large-scale maps. Existing SLAM solutions endow robots with capabilities to

accurately estimate maps and their positions in these maps [14, 15, 16, 17]. Yet, these solutions use representations that are not efficient. In terms of computation, many proposed algorithms either have fixed computational costs or require a complete set of sensed data. In terms of bandwidth usage, the typical point cloud representation [9, 8] requires large bandwidth budgets to share with other robots. These two key shortcomings, the high computational cost (often requiring GPU acceleration) and bandwidth requirement, prohibit SLAM from being used in robots that have limited computational or memory resources, e.g., light-duty UAVs and swarm-style robots. Hence, efficient use of computational resources and communication bandwidth is critical to deploying multi-agent systems.

Recent SLAM systems leverage deep learning frameworks to extract high-level information such as semantics for use in downstream algorithms, e.g., natural language processing [18, 19, 20, 21, 22]. However, these systems do not leverage semantic information to simplify the geometric representation of the sensed data or the SLAM map. This work investigates methods that promise to allow this information to be feedback into low-level processing functions such as relative pose estimation and depth map sharing, or higher-level tasking.

Objects with a complex pattern in the environment can be transformed into geometric primitives and can be also reconstructed from these primitives based on some visual rules, such as shape grammars [23], which can help address many of the aforementioned problems. With these modeling grammars being known by robots in advance, the robots are able to reconstruct the scene using very limited information like the model of objects (determined by semantics) and their locations with respect to the robots. Further, the grammatical description of the world makes it possible to extract abstract natural language (NL) descriptions of the world. For example, "there is a wall (associated with the shape grammars of the wall) behind a table (associated with the shape grammars of the table) at the position (x, y, z)". This level of NL

description not only permits natural-language-facilitated human-robot cooperation (NLC) but also provides a compressed description of the world. This allows closer collaboration between humans and robots which has received increasing attention in the recent decade [24]. By using NL, human intelligence at high-level task planning and robot physical capability at low-level task executions, such as force, precision, and speed, are combined to perform intuitive cooperation.

Our work takes the first step in the exploration towards the goal, to enable mobile robots to describe the world with compressed data and share maps with much less bandwidth requirement. As an initial step towards this broader goal, we limit the geometric primitives described above to planes to construct our SLAM system. We base our solution on DVO-SLAM [10] and our previous work, compute-bound and low-bandwidth RGB-D graph SLAM [25]. As a popular visual SLAM system, DVO performs well for RGB-D camera tracking and building accurate maps. It was extended in [25] with a fast plane fitting algorithm to extract surface information from scenes. A Quadtree [26] structure was also used to compress maps into a planar representation. However, this map representation is sparse and, thus, may contain large holes and ambiguous regions. Moreover, RGB-D data rather than plane data were used for odometry estimation, which lacked computation efficiency. Finally, neither RGB-D nor semantic information was available for representing the map (only geometries). In this article, we extend our previous framework for bandwidth and computation reduction with two novel depth compression algorithms and one improved independent plane fitting algorithm. These efficiently provide the system with dense plane fits. With the dense plane data available, we are able to utilize the fitted planes to calculate the odometry and then reconstruct dense maps. To provide robots with model information of the objects in the scene, we integrate a convolutional neural network (CNN) into the SLAM system to extract semantic information correlated to the planes in the environment. To summarize, the main contributions of our work

are as follows:

- Two efficient and effective compression algorithms for depth images are introduced and implemented;
- A real-time fast plane fitting method is proposed which can fit planes independently of the sensor intrinsic camera perimeters;
- A real-time odometry algorithm based on plane constraints is established;
- An RGB-D SLAM approach is developed which can construct and share 3D semantic maps with much less computational cost and bandwidth requirement;
- The potential is shown to estimate semantics from compressed geometric information by feeding planes to an RGB-D CNN for semantic segmentation;

These contributions significantly advance the state of the art for RGB-D SLAM. Taking advantage of one of the geometric primitives, planes, our method provides a new representation for point clouds that can be quickly calculated and represents the data to a similar degree of geometric accuracy using far fewer parameters. The joint effect of these contributions allows agents with 3D sensing capabilities to calculate and communicate compressed map information commensurate with their onboard computational and bandwidth resources. Our results show the ability of the proposed method to compress a depth image of 1MB in real-time to as little as 144KB. Plane fits can also be calculated independently in real-time which enables multiple agents to efficiently share the compressed map and build the map, which helps significantly reduce the bandwidth consumption. The results of our experiments also demonstrate that by using plane data for odometry, the computational cost can be saved by as many as 12 times. Additionally, our analysis of semantic segmentation results shows the potential and benefits of extracting semantic information from compressed geometry data. As a first step towards a generalized parametric/semantic model of the

world, our work motivates future research on SLAM where robots can estimate modeling information of objects from more shape primitives and share this understanding of the world with limited communication channel capacities.

The paper is organized as follows: Section 3.2 reviews prior work on saving computational cost and bandwidth for SLAM systems; Section 3.3 briefly discusses some background work that the proposed method is based on; Section 3.4 presents our solution; Section 3.5 analyzes the performance of the proposed method through an extensive evaluation. Finally, Section 3.6 concludes the paper.

3.2 Related Work

In this section, we provide an overview of prior work done on our contributions, focusing on how bandwidth usage is reduced and how computational cost is saved by state-of-the-art SLAM systems. We also review the currently used semantic segmentation neural networks in the field, to explore the use of deep learning techniques in interpreting geometric information.

3.2.1 Bandwidth of SLAM

Due to the bandwidth constraints and limited communication range, it is challenging to share large amounts of data among the agents in a distributed SLAM system [27]. To overcome this, Montijano et al. [28] propose a distributed communication network where every robot only exchanges the local matches with its neighbors. The algorithm propagates local to global contexts to solve for a global correspondence [29, 30] and manages to reduce bandwidth requirements by transmitting a subset of the map information as a collection of sparse features. Recent research has explored the use of compact geometric representations like planes to reduce the map size [31, 25, 32]. These sparse representations give rise to sparse map data that may however contain large holes and ambiguous regions. Renaud et al. [33] propose a multi-agent system that reduces the required communication bandwidth and com-

plexity by partitioning point clouds into parts. They then compactly describe each part using discriminating features to efficiently represent the environment. Cieslewski et al. [34, 35] minimize bandwidth usage by running place recognition on image frames and only sending the extracted feature vectors to the robots.

While all of these SLAM systems work efficiently in reducing bandwidth, these solutions suffer from multiple issues. They are either not designed for RGB-D data, or they do not provide dense 3D maps as part of their SLAM solution. Moreover, they may not be optimized for distributed multi-agent systems. Our bandwidth-reducing solution, in contrast, allows agents to fit planes to dense point clouds independently and to easily compare plane fits with each other. We leverage a dense planar representation of the world, as our first step to interpret the world with more geometric primitives.

3.2.2 Computational Cost of SLAM

Another major challenge for current visual 3D SLAM approaches is to overcome the burden that processing sensed RGB-D data places on the host's available computational resources. Centralized approaches [36, 37, 38] address computational cost by aggregating data from multiple robots at a central server having more computational resources where SLAM estimates can be calculated. Yet, such approaches are not viable for RGB-D data since sharing this data requires a prohibitively large network bandwidth. Further, this computational model does not scale as the number of robots increases. Lajoie et al. [27] solves the SLAM optimization problem via distributed computation approaches. In this context, robots utilize only local computation and communication to optimize the SLAM pose graph and estimate robot trajectories as well as environment maps. Another distributed mapping algorithm [39] optimizes the SLAM algorithm by sharing key informative features. Recent research [40, 41] has extended this implementation as a backend to larger SLAM solutions as a method to reduce the computational burden of solving multiple robot trajectories in multi-agent

systems.

There has also been significant interest in compact shape models to mitigate computational issues. Planes, for example, have been used in SLAM for efficient surface data representation that can be associated and integrated with low computational complexity. Many plane-involved SLAM solutions, however, either extend feature-based SLAM with the use of planar surfaces [42], or have an orthogonal assumption on the environment which is less applicable [43]. Salas-Moreno et al. [44] build a dense planar SLAM system using a dense ICP method to estimate sensor poses, which requires GPU for real-time computation. A quaternion-based minimal plane representation is utilized by Kaess et al. [45] to update the planes during optimization without using GPU but the system does not perform well in real-time. Real-time CPU-only execution of dense planar SLAM algorithm succeeds in exceeding current popular online 3D reconstruction methods in pose estimation [31], while the computational cost can be further saved by aligning planes for loop closures instead of searching for 3D point pairs.

While some of these solutions can be computation-efficient, they are either not applicable to 3D mapping scenarios in bandwidth-constrained contexts or not designed for multi-agent systems. In this article, to further reduce the computational complexity, we distribute the computation burden across the SLAM system, perform plane fitting to RGB-D data in real time, and utilize plane representation for odometry and map building.

3.2.3 Depth Compression

The compression of sensed raw data is also critical for distributed visual-SLAM systems to perform in bandwidth-constraint platforms. Shum et al. [46] presents a detailed summary of image-based representations of video, textures, depth and etc. Researchers have previously tried to apply color image-based compression techniques on depth images but, considering that depth images are significantly different from

color images, standard color compression techniques may not be optimal. Nevertheless, several lossy schemes which are based on color image-based compression have been proposed for compressing depth images. For example, Krishnamurthy et al. [47] use a JPEG2000-based technique to achieve an approximate compression ratio of 50× on depth images. Mehrotra et al. [48] present a lossless entropy encoding algorithm that stores the inverse depth values as integers. Wildeboer et al. [49] present an H. 264-based scheme to compress depth maps from a video sequence. Pratapa et al. [50] present a random-access depth compression algorithm that generates a compressed depth image by partitioning the scene into three parts and processing each part independently.

With the lack of study in this area, in this article, we introduce and implement two novel compression algorithms for depth images. The first algorithm is an implementation of the lossless data compression library *zlib* [51] on depth data, which uses a dictionary-based compression entropy encoding scheme. The second algorithm is a novel random access compression algorithm that implements the *zlib* algorithm on 8 × 8 blocks and is described in detail in Section 3.4.1.

3.2.4 Semantic Segmentation Neural Networks

Assigning meaningful semantic labels to objects on the map is a non-trivial task and there exists a great deal of prior work on the subject. Some of the many approaches to this are presented here for context. The first design decision is whether to generate many labels by performing semantic segmentation or fewer with object detection/recognition. The difference is that object detection/recognition attempts to assign a label to an entire image or sub-image. Alternatively, semantic segmentation attempts to assign a label to each pixel. There are advantages and disadvantages to each approach, but in this work, we are interested in making dense planar maps, so we utilize semantic segmentation.

In the context of machine-learned semantic segmentation, the most well-known

semantic segmentation network is Mask R-CNN [52] which is a large network that is available pre-trained on a large dataset of images of everyday objects in context. Unfortunately, the objects it is trained on are poorly represented by planes and also generally make poor landmarks. Instead, we use a modified version of the RedNet [53] architecture. RedNet differs from Mask R-CNN in that it was intended to be used for indoor navigation and as such has two branches, one for RGB images and one for depth images. These branches extract features from their respective inputs and then merge the two data streams to perform the segmentation. This has the advantage that the depth images are more useful for segmentation and extracting object edges while RGB images contain more information for object recognition. We modify the network to use plane coefficient images and retrain on the same the SUN RGB-D [54] dataset but with the depth images pre-processed into plane coefficient images. As will be discussed in the Methodology section, operating on plane images provides an inductive bias that allows for certain classes to be more easily segmented.

3.2.5 RGB-D SLAM

We choose some representative state-of-the-art RGB-D systems and make a comparison in Table 3.1. Ours particularly stands out in providing bandwidth reduction and semantic information. Additionally, in contrast to [10, 55, 56, 45], our SLAM system has frontend-backend separability. This allows the system to distribute computation tasks on different CPUs or on completely distinct connected hosts, from which a multi-agent system can benefit. Note that the discussion of bandwidth is not applicable to systems that cannot be directly deployed in a distributed network, as those systems only run on a single host. All of the mentioned SLAM systems can run in real-time according to their articles while some of them require GPU acceleration. As shown in the table, we are working on a new space in terms of bandwidth reduction and a semantic world, which makes a fair comparison difficult. The main purpose of this article is to provide different solutions to the ultimate goal which is

making robots understand and communicate the knowledge of the world with limited onboard resources.

Table 3.1: Comparison of selected existing RGB-D SLAM with our SLAM system.

	Frontend -backend Separability	**Bandwidth Reduction**	**CPU only**	**Local Mapping**	**Semantic**
DVO [10]		-	✓	Point-Based	
ORB-SLAM2 [9]		-	✓	Point-Based	
Bundle Fusion [55]		-		Volumetric	
BAD-SLAM [56]	✓			Point-Based	
Dense Planar SLAM [44]		-		Plane-Based	
SLAM w/ Infinite Planes[45]		-	✓	Plane-Based	
Point-Plane SLAM [57]	✓		✓	Plane-Based	
Ours	✓	✓	✓	Plane-Based	✓

3.3 Background

In this section, we provide background on key concepts of the planar semantic SLAM which were discussed in detail in our previous publications. We will specifically focus on giving readers background on graph SLAM and the plane fitting algorithm previously described in [25].

3.3.1 Graph SLAM

Building a factor graph representation of a robot's state is a popular technique for solving the SLAM problem [58, 59, 60, 61, 62, 63, 64]. Factor graphs use a sparse representation of the robot's states to estimate the *full* trajectory of a robot and are ideal for compute-bound systems. The graph SLAM problem can be formulated as estimating the posterior probability shown in Equation (3.1), where a graph is constructed as the robot moves through an unknown environment map $\mathbf{m}$ along a trajectory represented by the sequence of random variables $\mathbf{x}_{1:T} = \{\mathbf{x}_1, ..., \mathbf{x}_T\}$. While moving from an initial position x_o, the robot acquires a sequence of odometry measurements $\mathbf{u}_{1:T} = \{\mathbf{u}_1, ..., \mathbf{u}_T\}$ and perceptions of the environment $\mathbf{z}_{1:T} = \{\mathbf{z}_1, .., \mathbf{z}_T\}$.

$$p(\mathbf{x}_{1:T}, \mathbf{m} | \mathbf{u}_{1:T}, \mathbf{z}_{1:T}, x_o) \tag{3.1}$$

In a graph-based SLAM approach, the robot poses are represented as nodes/vertices which relate to their position in the environment. The spatial constraint between vertices are obtained from either the odometry measurements $\mathbf{u_t}$ or sensor measurements $\mathbf{z_t}$ and are represented as edges. An edge constraint is obtained either between two consecutive robot positions or by aligning sensor observations between two robot locations (loop closures).

Solving the posterior leads to an optimization problem over a sum of nonlinear quadratic constraints in the graph. Once the graph is constructed, we seek the configuration of the graph vertices that best satisfy the set of constraints and we seek a Gaussian approximation of the posterior distribution of Equation (3.1). The optimal trajectory, $\mathbf{x}^\star$, is estimated by minimizing the joint log-likelihood of all constraints as described by Equation (3.2) [65].

$$\begin{aligned}\mathbf{x}^\star = \min_{\mathbf{x}}(x_o^T\Omega_o x_o + \sum_t [x_t - g(u_t, x_{t-1})]^T R_t^{-1}[x_t - g(u_t, x_{t-1})] + \\ \sum_t [z_t - h(x_t, m_i)]^T Q_t^{-1}[z_t - h(x_t, m_i)])\end{aligned} \tag{3.2}$$

The leftmost term, $x_o^T\Omega_o x_o$, represents our prior knowledge of the initial pose where Ω_o is the inverse covariance matrix associated with the initial motion. Often, x_o is set to zero, anchoring the map to the origin. The middle term describes a sum over the motion constraints. The residual vector $x_t - g(u_t, x_{t-1})$, is then the difference between the expected pose, x_t, and the pose observed by applying the motion estimate $g(u_t, x_{t-1})$. Similarly, The rightmost term describes a sum over landmark constraints. The residual vector $z_t - h(x_t, m_i)$, is then the difference between the expected landmark pose in the global coordinate system, and the estimated global landmark pose resulting from the local landmark measurement. R_t^{-1} and Q_t^{-1} are the information matrices or inverse covariance matrices associated with the motion constraint and the landmark constraint respectively.

As mentioned earlier, the minimization is commonly solved using nonlinear optimization approaches like, e.g., Levenberg-Marquardt, Gauss-Newton, etc. It can also be solved efficiently by exploiting the sparse structure of the graph SLAM formulation [66], thereby allowing the use of sparse methods such as sparse Cholesky decomposition or the method of the conjugate gradient. Many graph optimization libraries, such as the g^2o library [67, 68], leverage these sparse methods in order to quickly optimize large graphs.

Many SLAM systems can be well described in terms of two fundamental components: (1) the frontend and (2) the backend. The frontend tracks camera pose changes from RGB-D images by minimizing the reprojection errors between two image pairs, which results in a delta-pose measurement of the camera's ego-motion and associated uncertainty. The backend is responsible for large-scale map integration. The estimated camera poses and their uncertainties are used to establish a vertex obtained from keyframe data. Edges connect vertices using sensor data from connected keyframes. When two keyframes include overlapping views of the same geographic map regions, the graph creates a new constraint also called loop-closures.

One important visual SLAM implementation is Direct Visual Odometry (DVO) [10]. DVO implements a keyframe-based approach for map building where the visual odometry algorithm uses an RGB-D pair (keyframe) and estimates odometry between this frame and subsequent frames. Once a certain threshold such as distance, bearing change, or uncertainty, a new keyframe is established. The 3D map which is constructed on the backend is a collection of the keyframe pose at the time it was established.

Our previous work [25] extends DVO SLAM by adding several new capabilities. One of the most significant new capabilities allows separates the frontend, and backend components of this system into distinct applications which share information via network communication. These components can run on the same or different CPUs

or on completely distinct hosts connected over a low-bandwidth network.

3.3.2 Real-time Plane Fitting to RGB-D data

In [25], we seek to approximate the measured depth data with 3D planar surfaces. The planar representation for 3D scenes promises to significantly reduce the size of the RGB-D image data which serves to reduce both memory usage and computational costs. Plane fitting is accomplished in real-time using an ultra-fast-fitting algorithm proposed in [69]. This accelerated fitting of planes is made possible by a rearrangement of the standard plane fitting error functions for RGB-D data. Standard planar representations adopt a form of equation $aX+bY+cZ+d=0$ where $X = Z*\left(\frac{x-c_x}{f_x}\right)$ and $Y = Z*\left(\frac{y-c_y}{f_y}\right)$ and $Z = Z(x,y)$, where (x,y) is the image pixel coordinate and (c_x, c_y, f_x, f_y) are camera intrinsic parameters. We substitute the 3D reconstruction equations for the variables X and Y then simplify the resulting equation to fit directly to only the measured RGB-D depths and, in doing so, save significant computational cost.

The re-arrangement of the terms gives

$$\frac{aX}{dZ} + \frac{bY}{dZ} + \frac{c}{d} + \frac{1}{Z} = 0 \tag{3.3}$$

In order to fit planar models to sets of 3D points, we leverage an explicit least-squares formulation to compute the coefficients of the plane as shown by the objective function in the equation. We rename the variables as follows $\alpha_1 = \frac{a}{d}$, $\alpha_2 = \frac{b}{d}$ and $\alpha_3 = \frac{c}{d}$ and solve the explicit least-squares fitting problem in Equation (3.4) below.

$$f(x,y) = \sum_{(x,y)} \left\| \alpha_1 \left(\frac{x-c_x}{f_x}\right) + \alpha_2 \left(\frac{y-c_y}{f_y}\right) + \alpha_3 + \frac{1}{Z} \right\|^2 \tag{3.4}$$

Note that $\frac{x-c_x}{f_x}$ and $\frac{y-c_y}{f_y}$ can be pre-computed from the known integer (x,y) pixel coordinates and the calibration parameters.

This equation provides us the benefit of conceiving a computational algorithm that will fit an explicit 3D planar surface directly to the measured, i.e., perspective-projected, N depth image values. The result of the fit depends on the camera intrinsic parameters (f_x, f_y, c_x, c_y). The least-squares scatter matrix for this fitting approach has the form as shown in Equation (3.5) below.

$$\mathbf{M}^t\mathbf{M} = \sum_{i=1}^{N} \begin{bmatrix} \left(\frac{x_i - c_x}{f_x}\right)^2 & \left(\frac{x_i - c_x}{f_x}\right)\left(\frac{y_i - c_y}{f_y}\right) & \left(\frac{x_i - c_x}{f_x}\right) \\ \left(\frac{x_i - c_x}{f_x}\right)\left(\frac{y_i - c_y}{f_y}\right) & \left(\frac{y_i - c_y}{f_y}\right)^2 & \left(\frac{y_i - c_y}{f_y}\right) \\ \left(\frac{x_i - c_x}{f_x}\right) & \left(\frac{y_i - c_y}{f_y}\right) & 1 \end{bmatrix} \tag{3.5}$$

where $\mathbf{M}$ denotes the matrix of planar monomials formed from the 3D (X, Y, Z) surface data having i^{th} row $\mathbf{M}_i = \begin{bmatrix} \frac{x_i - c_x}{f_x} & \frac{y_i - c_y}{f_y} & 1 \end{bmatrix}$. The least squares solution for the unknown vector $\boldsymbol{\alpha}^t = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 \end{bmatrix}$ is then $\boldsymbol{\alpha} = (\mathbf{M}^t\mathbf{M})^{-1}\mathbf{M}^t\mathbf{b}$ where $\mathbf{b}$ is the inverse of the measured, i.e., perspective projected, depths; $\mathbf{b} = \begin{bmatrix} \frac{1}{Z_0} & \cdots & \frac{1}{Z_N} \end{bmatrix}$ which is proportional to the pixel disparity typically used for depth calculation in depth sensors [70].

It is important to note that none of the elements of the scatter matrix depend on the measured depth data and, as such, this matrix requires a constant number of operations to compute for each measured image, i.e., it can be pre-computed given the RGB-D camera parameters. Hence, explicit plane fitting in the RGB-D range space requires the only computation of the vector $\mathbf{b} = \begin{bmatrix} \frac{1}{Z_0} & \cdots & \frac{1}{Z_N} \end{bmatrix}$ for each range image and the best-fit plane is given by a single matrix multiplication: $(\mathbf{M}^t\mathbf{M})^{-1}\mathbf{M}^t\mathbf{b}$, where the value of $\mathbf{M}^t\mathbf{b}$ is given below:

$$\mathbf{M}^t\mathbf{b} = \sum_{i=1}^{N} \frac{1}{Z_i} \begin{bmatrix} \frac{x_i - c_x}{f_x} \\ \frac{y_i - c_y}{f_y} \\ 1 \end{bmatrix} \tag{3.6}$$

3.4 Methodology

The goal of the proposed approach is to efficiently and effectively compress the knowledge of the world. This enables robots with limited resources to contribute to and potentially benefit from 3D maps in a distributed 3D SLAM system. This goal is achieved by the following modifications to the previous work [25].

- Independent fast plane fitting - a coordinate transformation is applied so that solving the least-squares fitting problem is independent of the sensor's intrinsic parameters.
- Compression algorithm - a bitmap is used to record NaN value locations before compressing the depth and then encoding the depth with a custom dictionary-based compression algorithm.
- Odometry algorithm with planes - camera transforms are calculated by aligning the compact plane fits of the point cloud instead of repetitively matching every point.
- Semantic maps - plane images are populated to an RGB-D CNN for semantic segmentation as input to show how geometries help extract semantics.

The cumulative effect of these modifications generates a new SLAM system that advances the state-of-the-art for efficient multi-agent map building. The framework of the overall SLAM system is shown in Fig. 3.1, and it contains two parts: (1) the frontend where the depth data is compressed and the camera pose is estimated, and (2) the backend where the global semantic planar map is generated.

3.4.1 Depth Compression

When building multi-agent distributed SLAM systems, data compression plays an important role in reducing the bandwidth budget of agents, especially in Visual SLAM

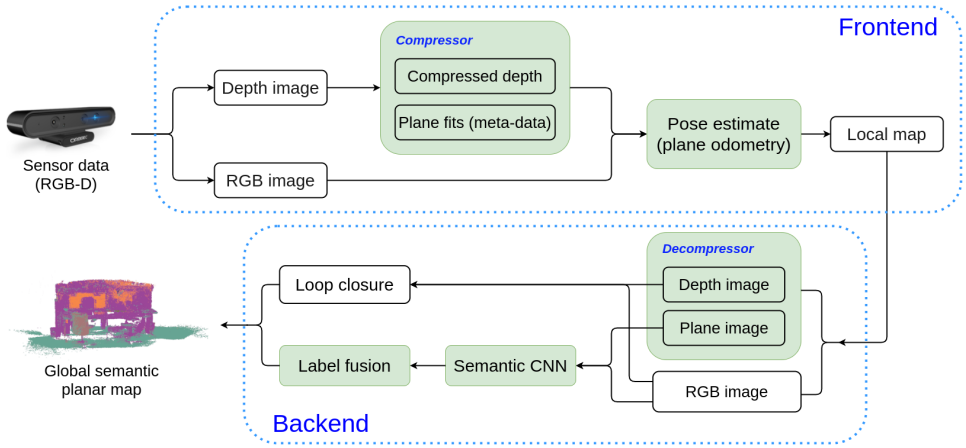


Figure 3.1: Framework of our low-bandwidth 3D planar semantic SLAM system. The frontend takes sensor data as input and then performs depth compression (Section 3.4.1) and plane fitting (Section 3.4.2) at the same time. The plane coefficients and RGB images are used for fast camera pose estimation (Section 3.4.3). Local maps contain all image and plane data, as well as camera pose and camera parameters. The backend takes as input local maps to recover the RGB, depth, and plane images (an image of plane coefficients), and create a SLAM graph with loop closures. The semantics are predicted by a CNN and integrated into the graph to generate a global map. Note that in our experiments the loop closure detection uses depth and RGB images to perform the more accurate mapping. However, for fast loop closure detection, the depth data can be replaced with plane data.

applications that use RGB-D sensors. While there are very well-established and highly-tuned algorithms to compress RGB data, not many algorithms are available for depth image compression. Several lossy schemes that work for standard color images have been used to compress depth images but do not perform as well. This is because the depth images have different properties than standard color images so the compression algorithms have unique needs to compensate for *NaN* values scattered through the image. This prevents typical grid-based algorithms for compression like the Discrete Cosine Transform (DCT) in JPEG compression, or the wavelet transform in JPEG 2000. To overcome these challenges, we propose two novel depth compression techniques, (1) a *zlib* based entropy encoding technique and (2) a custom dictionary-based compression approach that allows random access.

In our first approach, we leverage *zlib* library for lossless compression of depth im-

ages. *zlib* is a free, open-source library that is used in thousands of applications for data compression like compressing TLS connections and storing Git version control files. However, to the best of our knowledge, no previous work has applied it to depth compression. *zlib* library uses a deflate-inflate method to encode and decode data which in turn uses a combination of the Lempel-Ziv-Storer-Szymanski (LZSS) algorithm [71] and Huffman coding [72]. The LZSS is a dictionary-based algorithm that replaces recurring bytes of data with a reference to a previously occurred byte. The algorithm uses a sliding window-based approach to find sequences of repeated data. Then the Huffman encoding breaks LZSS encoded data into blocks and generates codes for each data block. The Huffman encoding uses a statistic-based approach to encode symbols whose lengths are based on the frequencies of occurrence. The inflate method follows similarly in reverse.

In our custom dictionary-based approach, we adopt the concept of the bitmap to locate the pixels with invalid depth (*NaN* values). We divide a 640x480 depth image into multiple 8x8 blocks. For each block, we create a bitmap to denote if an invalid pixel is existent. Specifically, in this bit pattern, for each location, if the pixel contains a *NaN* value, the bit value is set to 1. Essentially we create a bitmap representation of the block where 1 indicates where the *NaNs* are and 0 indicates where the depth measurements are. This representation requires 64 bits (8 bytes) to map the location of *NaNs*. Having all *NaN* values located, we traverse the bitmap in a zigzag pattern to convert the 2D data to a 1D vector while jumping over all of the *NaN* values that are possibly scattered in the bitmap. We then compute a compressed bitmap buffer of these *NaN* pixel locations and encode it with run-length encoding (RLE). The *non-NaN* values are encoded by a dictionary-based lossless compression algorithm using *zlib*. By filtering invalid pixels beforehand and respectively encoding the *NaN* value locations and valid depth information, we are able to compress the geometric data while avoiding the potential problems of direct depth compression. To decompress

the data, the algorithm decodes the *NaN* bitmap and recovers the dictionary used in the encoding stage to decode the values. The steps for both encoding and decoding are summarized in Fig. 3.2.

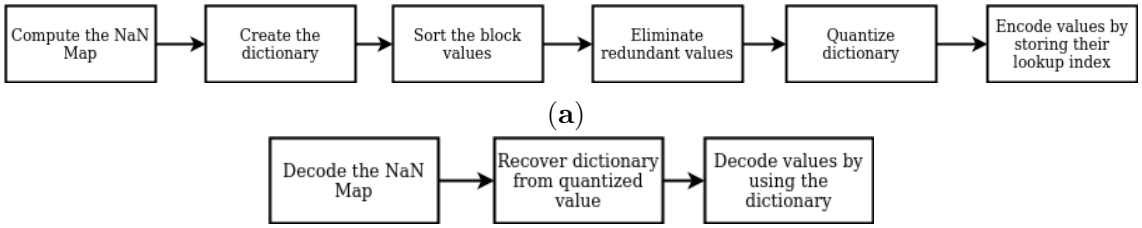


Figure 3.2: A block diagram summarizing the steps involved in encoding and decoding the depth map in the custom depth compression algorithm.

3.4.2 Stream Meta Data

On top of our compressor, we can optionally add a layer of surface information metadata by fitting planes to 8x8 blocks of depth data. For each block of the depth image, if the block contains sufficient valid depth data (more than 50% pixels), we apply the surface fitting algorithm mentioned in Section 3.4.2.1 to calculate a surface representation and an entity of metadata, including the plane coefficient vector and its covariance that summarizes the log-likelihood of the pixel in that block given the plane fit.

3.4.2.1 Independent Plane Fitting

To approximate the measured depth data with a 3D planar surface, we leverage the plane fitting algorithm discussed in Section 3.3.2 and extend it to serve better in multi-agent scenarios. We devise a method to fit explicit planar surfaces to local image patches without the need to know the location of the image center. This allows local $N \times N$ image patches to be compressed without the knowledge of their absolute location in the image. Let (x, y) denote the pixel coordinates within an image region bounded by upper-left coordinate (x_0, y_0) and lower right coordinate (x_1, y_1). We denote the patch center as the point $(x_c, y_c) = (\frac{x_1 - x_0}{2}, \frac{y_1 - y_0}{2})$. We then make the

coordinate transformation in Equation (3.7).

$$\begin{aligned} x_i &= x_i' - x_c + c_x \\ y_i &= y_i' - y_c + c_y \end{aligned} \tag{3.7}$$

The impact of this coordinate transformation changes the scatter matrix $\mathbf{M}^t\mathbf{M}$ from the Equation (3.5) to the Equation (3.8) in (x', y') shown below.

$$\mathbf{M}^t\mathbf{M} = \sum_{i=1}^{N} \begin{bmatrix} \frac{(x_i'-x_c)^2}{f_x^2} & \frac{(x_i'-x_c)(y_i'-y_c)}{f_x f_y} & \frac{(x_i'-x_c)}{f_x} \\ \frac{(x_i'-x_c)(y_i'-y_c)}{f_x f_y} & \frac{(y_i'-y_c)^2}{f_y^2} & \frac{(y_i'-y_c)}{f_y} \\ \frac{(x_i'-x_c)}{f_x} & \frac{(y_i'-y_c)}{f_y} & 1 \end{bmatrix} \tag{3.8}$$

Equations formulated in the coordinate system (x', y') enjoy the benefit of being independent of the sensor intrinsic camera parameters (c_x, c_y), and require only the knowledge of the size of the image patch, i.e., (x_0, y_0) and (x_1, y_1). Calculation of surface fits at the coordinates (x', y') can then be linearly put into the 3D coordinate system appropriate for a sensing camera by a single linear transformation on the estimated variable $\boldsymbol{\alpha}$ as shown in Equation (3.9) below.

$$\mathbf{T}_{\boldsymbol{\alpha}'} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \frac{c_x-x_c}{f_x} & \frac{c_y-y_c}{f_y} & 1 \end{bmatrix} \tag{3.9}$$

This allows inverse values for the matrix $\mathbf{M}^t\mathbf{M}$ to be pre-computed and enables the resulting fit surfaces to be placed in 3D with a single linear transformation of the estimated coefficient vector: $\boldsymbol{\alpha} = \mathbf{T}_{\boldsymbol{\alpha}'}\boldsymbol{\alpha}'$.

Various applications including multi-agent SLAM can be benefited from this separation of the plane fitting algorithm and the sensor information. In a multi-agent SLAM system where agents are likely equipped with different cameras, our camera-independent fitting algorithm allows the possibility of easily sharing and comparing

plane fits across the cameras which enables an efficient cooperative SLAM system.

In addition to encoding the depth map, our compression algorithm also encodes the camera calibration information at the same time. After compressing all information, we send from frontend to backend the compressed depth image, the plane fits each of which takes three 24-bit values and plane coefficient covariance (six 24-bit values) for all blocks having 32 or more valid depths in an 8x8 window.

3.4.3 Plane Cloud Odometry

Similar to DVO SLAM, we use alignment results between images for odometry. Given an image pair with intensity and depth information (I_1, Z_1) and a second image pair taken at a later time (I_2, Z_2), DVO odometry algorithm seeks to estimate the set of transformation parameters ξ_e, that best aligns the images after a warp function $\tau(\mathbf{x}, T)$ is applied to all pixels $\mathbf{x}$ where T is the transformation matrix of the camera motion and can be calculated from ξ_e using the matrix exponential $\mathbf{T} = \exp(\xi_e)$. The warped location in the second image, $\mathbf{x}'$, of an image pixel $\mathbf{x}$ in the first image can be computed given the transformation matrix, T, and the projection function, π, from pixel coordinates to a 3D point.

$$\mathbf{x}' = \tau(\mathbf{x}, \mathbf{T}) = \pi(\mathbf{T}\pi^{-1}(\mathbf{x}, Z_1(\mathbf{x}))) \tag{3.10}$$

The goal here is to find the best correspondence that is determined by the difference between the warped image, $W(\mathbf{x}')$, and the reference image, $I(\mathbf{x})$ or $Z(\mathbf{x})$ in terms of intensity and depth value. The error function can be written as:

$$f(\xi) = \frac{1}{2}\sum_{\mathbf{x}}(W_I(\mathbf{x}') - I(\mathbf{x}))^2 + \frac{1}{2}\sum_{\mathbf{x}}(W_Z(\mathbf{x}') - Z(\mathbf{x}))^2 = \frac{1}{2}\sum_{\mathbf{x}} r_I(\mathbf{x})^2 + \frac{1}{2}\sum_{\mathbf{x}} r_Z(\mathbf{x})^2 \tag{3.11}$$

Note that the error term above is nonlinear, and we, therefore, find ourselves in the

midst of a nonlinear least-squares problem. It is worth mentioning that a multi-scale methodology is adopted by DVO for odometry calculation. At different levels, it first computes the residuals for intensity and depth, then computes the sum of residual weighted gradient vectors for parameter update.

The problem of matching image measurements (depth and intensity) is that it requires the existence of the same sample from the same surfaces, which is not always available with the camera moving. The common solution to this problem is to perform interpolation which introduces much computation to the problem. For example, DVO Odometry performs interpolations for depth images, intensity images, depth gradients in x, y directions, and intensity gradients in x, y directions at every iteration, as a result of which, it is very computationally expensive. This can be addressed by directly solving for the correspondences between plane fits from two plane images [32]. We use the compact representation of the world in terms of planar algebraic surfaces, i.e., surfaces having equation $ax + by + cz + d = 0$, to establish the likelihood that a given hypothesized plane image pair can be aligned with the intent to piece together large geometric map regions in a manner similar to puzzle-solving. This is accomplished by minimizing an error function that solves for both the correspondence of planar surfaces between the plane images and the Euclidean transform that aligns these algebraic surfaces. The magnitude of the algebraic alignment error then serves as a goodness-of-fit metric to validate or refute the plane image pair hypotheses. We base our Plane Odometry algorithm on this and it saves much computational cost by avoiding searching for corresponding measurements from surfaces at the corresponding location. Instead, it computes the optimal alignment (in the least-squares sense) between planes that are hypothesized to have the same equation up to an unknown Euclidean transformation. For our derivation, we denote π_j and π_l as two collections of N plane equations. Equation (3.12) expresses the optimization problem at hand. Here we seek to estimate the transformation $\widehat{\mathbf{T}}_{i\to j}$ that takes the planes of π_l into the

coordinate system of planes π_j. Note that Euclidean transformations, when applied to planes, follow the transformation rule $\pi^{'} = (\mathbf{T}^{-1})^t \pi$ for $\pi^t = [a, b, c, d]$.

$$\widehat{\mathbf{T}}_{i \to j} = \min_{\mathbf{T}_{i \to j}} \sum_{\{i,j\} pairs} |0\pi_j - (\mathbf{T}_{i,j}^{-1})^t \pi_i|0^2 \tag{3.12}$$

To benefit from both image measurements and algebraic representation of points, we can alternatively add a stream of metadata from our plane fitting on top of the intensity constraints in Equation 3.11, in replace of the depth constraints. The new error function is shown in Equation 3.13. We call this approach Hybrid Odometry. Specifically, our Hybrid Odometry treats plane coefficients (a, b, c, d) as four images in addition to the intensity image that is already used by DVO. The transformation to be solved needs to bring 4 plane coefficients and intensity values into an agreement between two frames. Intensity images are required to be downsampled to match the size of plane images and interpolations are performed during calculation. By taking advantage of the metadata within blocks, our method aligns images by performing calculations on a much smaller collection of values instead of visiting every pixel in the image.

$$f(\xi) = \frac{1}{2} \sum_{\mathbf{x}} r_I(\mathbf{x})^2 + \frac{1}{2} \sum_{\mathbf{x}} r_P(\mathbf{x})^2 \tag{3.13}$$

We also perform graph-SLAM incorporating only motion constraints into the back-end optimization like DVO SLAM. However, in contrast to DVO SLAM, our backend keyframes store compressed plane clouds in lieu of RGB-D data. This fundamentally changes the function $g(u_t, x_{t-1})$ in the second term of the graph-SLAM optimization problem in Equation (3.2) from an RGB-D point cloud alignment problem to that of aligning compressed plane clouds. The new optimal pairwise motion estimate for a keyframe pair is replaced by a new odometry algorithm $g(u_t, x_{t-1})$ that estimates the relative motion of a pair of point clouds by minimizing the corresponding pairwise

plane cloud data error.

3.4.4 Semantic SLAM

Our semantic slam approach applies a CNN that takes in RGB images and plane images which are sets of plane coefficients associated with each pixel, to produce semantic labels for each input pixel. Once semantic labels are assigned to each element label fusion is used to ensure that labels are consistent across time and multiple views. These are the core components of semantic slam: semantic segmentation to generate labels and label fusion to combine and track them. These are discussed in the following subsections.

3.4.4.1 Plane Semantic Segmentation Network (RedNet)

We use RGB + Plane coefficient data as the inputs into a multi-branch convolutional neural network that performs semantic segmentation. Semantic segmentation is a combination of image segmentation, in which pixels correspond to the same object and object classification, identifying and labeling that object. This is useful both for producing labeled maps and for reducing the computational burden of identifying correspondences and loop closures as there is no sense in trying to match a table to a wall.

There are many networks, such as RedNet [53] and BiSeNEt [73] that use both RGB and depth information to perform semantic segmentation. These networks generally perform convolutional operations on the RGB and depth images in parallel and then fuse the information together before scaling back up to the original resolution, though implementation details differ. We are forced to retrain on a targeted dataset with modifications to the depth branch to operate on plane images.

A plane image is similar to a depth image except it contains plane coefficients instead of depth values. Since a plane can be represented with as few as 4 coefficients, we are able to reconstruct 3D geometric data into a 4-channel image but with 3D

information for the following semantic segmentation work. In a real-world scene, especially an indoor scene, many of the objects can be fitted by planes, such as floors, walls, and tables. For this work, we operate on plane coefficients but do not merge contiguous planes so as to keep the convolutional architecture. Operating on irregular collections of planes is deferred to future work.

One advantage of performing the segmentation on planes rather than on the depth values is that depth values are independent but the plane coefficients encode information about the local geometry. Based on preliminary results, we elected to focus on modifying and retraining RedNet for our application. RedNet was originally trained on RGB+D data from the SUN RGB-D dataset [54]. We train our whole network on a modified version of the same dataset in which the depth channel has been converted to a grid of plane coefficients, i.e., plane images and the RGB images scaled to align with the plane images.

We reused the PyTorch implementation of RedNet and retrained both branches with the depth branch converted to use plane images in the place of depth images. Both branches are downsampled by necessity as calculating each plane requires at least 3 valid depth points within one fitting block. No pre-trained weights for the RGB branch are available, but we follow the training regimen specified in the original RedNet paper.

3.4.4.2 Label Fusion

Generating a semantic map requires the temporal consistency of the semantic prediction. When observing a scene from a moving camera such as on a mobile robot, the system obtains multiple different views of the same objects. Nevertheless, as the viewpoint varies, different semantic cues estimated by the CNN may become available and a previously semantically ambiguous region may become more distinctive. To address this problem, we perform data association by warping sequential frames of different views into a common reference view and fusing the semantics.

As discussed in Section 3.4.3, given 2D image coordinate $\mathbf{x} \in \mathbb{R}^2$, the warped pixel location can be determined by Equation (3.10). With the warped pixel location, we can find the pixel correspondences for two label images S_A, S_B in sequential keyframes sharing a common field of view (FOV). We then compare the labels and their associated prediction confidence for those pixels in the common FOV. If the labels are the same for two corresponding pixels, we remain the same label in the label image of the second view and update the predicted confidence by averaging the two confidence. If not, we update the pixel label in both label images with the label of higher confidence. The new confidence is obtained by lowering the higher confidence by 10% as a penalty for semantic inconsistency.

The underlying intuition of our label fusion is that corresponding pixels must have the same semantic label, as well as similar (but not necessarily the same) prediction confidence. Unlike the photo-consistency assumption adopted by tracking algorithms like DVO SLAM [10], the semantic consistency assumption is comparatively weak since it is not anchored to any actual measurement. However, it is possible to use it as a constraint for graph optimization. In this work, we only focus on using label fusion to generate a global semantic map, not on any optimization.

3.5 Results

In this section, we evaluate the key components of our SLAM system that we contribute to. We conduct experiments on each component separately and compare it to the state of the art when available. Specifically, (1) we compare and analyze the size of our compression data to the raw data, showing bandwidth reduction in sharing the geometry information among agents. (2) We also compare our odometry algorithms with a popular RGB-D camera tracking algorithm DVO Odometry, demonstrating that using plane data accelerates the odometry estimation process by reducing the data size and calculation complexity. (3) We then evaluate the performance of our CNN with a common RGB-D semantic segmentation CNN, RedNet. Our results

outperforming RedNet on some classes suggest the potential of extracting semantics from plane data in an indoor scene where most objects have regular shapes. The TUM RGB-D dataset [74] is used for evaluation and comparison for most of the experiments while the neural network training uses the SUN-RGBD dataset. More details are discussed in the following sections respectively.

3.5.1 Depth Compression

Depth compression for SLAM is a relatively new concept and to the best of our knowledge, there exists no open-source implementation of compression on non-synthetic depth data. In this work, we present two novel depth compression techniques, *zlib* Compression and the UNCC Compression technique. We conduct our experiments on a laptop with an Intel Core i9 processor (no GPU used). We evaluate the performance of the two algorithms by examining the time taken to encode and decode a depth image and by investigating the size of the compressed depth map. Fig. 3.3a shows the size of the compressed depth map for each of the 2510 depth images from a benchmark TUM RGB-D dataset. Table 3.2 provides statistics on the size of the compressed depth images for the same experiment. It is observed that both the *zlib* and UNCC Compression have a compression ratio of 9.6× and 8.5× respectively. As expected, the UNCC compression algorithm has a larger compressed image, as it encodes the compressed image with meta-data containing the plane coefficients. Additionally, each 8×8 block has its own dictionary taking up more space.

As described in the previous section, the UNCC compression algorithm splits an image into 8×8 blocks and encodes the non-NaN values using a dictionary generated by the *zlib* algorithm, which means that each 8×8 block has a dictionary that needs to be encoded. This gives the user flexibility in the decoding process in terms of decoding only parts of the image and parallelizing the decoding process. In addition to the depth map, the UNCC compression technique also encodes the camera calibration information and the plane fits for each block, thus taking up more time and size to

encode a depth map. Upon further investigation, the average size difference for the encoded depth map produced by the *zlib* algorithm and the UNCC compression is ∼17kB.

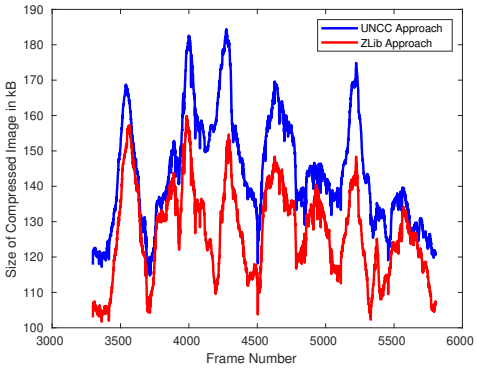
Table 3.2: Statistics of the compressed image size

Compression	***zlib* Compression**	**UNCC Compression**	**Original Size**
Mean [kB]	127.16	144.24	1228.8
Std Dev. [kB]	12.99	15.74	N/A
Max [kB]	159.84	184.32	N/A
Min [kB]	101.72	114.76	N/A

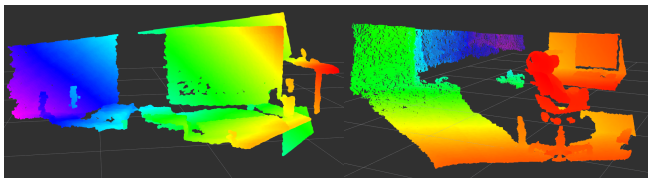
Fig. 3.3a shows the size of the compressed depth image obtained from both the *zlib* and UNCC algorithms for each frame in the test dataset. Depth sensors perform better when objects are near the camera when compared to objects that are further away. This means that they encode more information on nearby objects producing high-resolution maps on scenes with lots of objects near the cameras like the one shown in Fig. 3.3b as compared to the scene shown in Fig. 3.3c. As the amount of information encoded in the original depth image is high, it results in a larger compressed image, which is seen in Fig. 3.3a. Fig. 3.3b and 3.3c are the 4000^{th} and 4500^{th} frames in the dataset respectively.

We show the statistics for the time taken to encode and decode depth images for the two compression algorithms in Table 3.3. The time taken to decode the compressed depth image is significantly lower than the time taken to encode the data, especially for the UNCC compression technique. This is expected because to encode the depth map, the algorithm has to sort values in an 8×8 block and then remove redundant values to generate the dictionary, which is not necessary for decoding. The small duration to decode the depth image and the small size of the compressed image opens the door to storing the depth image in a compressed format. This frees up storage space for running the algorithm for longer and saves more data in memory.

In addition, the ability of the UNCC algorithm to decode blocks individually allows



(**a**)



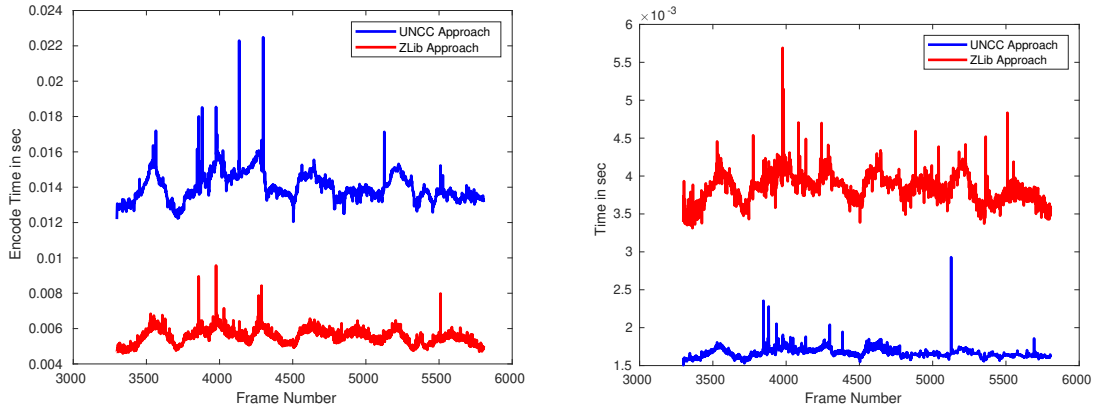
(**b**) (**c**)

Figure 3.3: (a) The size of the compressed depth image (in kB) for each frame in depth dataset. (b) The depth point cloud capture at frame 4000. The depth image consists of objects near the camera thus containing more information to encode (as seen in the plot). (c) The depth point cloud at frame 4500. The majority of the objects captured are relatively far from the sensor, resulting in a low-resolution depth image thus resulting in a smaller compressed depth image.

the reconstruction of the depth map by sharing only the plane metadata between the frontend and the backend. It is noted that it takes an average of $1.24\mu s$ to decode a single block from the compressed depth image which is comparable to Pratapa et al. [50] who present a random access compression technique on individual depth frames. This allows the possibility to further decrease the bandwidth of the computation load on both the frontend and the backend in future work.

Table 3.3: Statistics of the time required to encode and decode depth images for two compression algorithms.

	***zlib* Compression**			**UNCC Compression**		
	Encode	**Decode**	**Total**	**Encode**	**Decode**	**Total**
Mean [μs]	5544.89	3844.98	9389.87	13997	1665.93	15663.26
SD [μs]	433.74	191.97	601.79	818.75	62.68	869.15
Max [μs]	9568	5691	15259	22485	2930	24524
Min [μs]	4590	3312	7944	12044	1505	13592

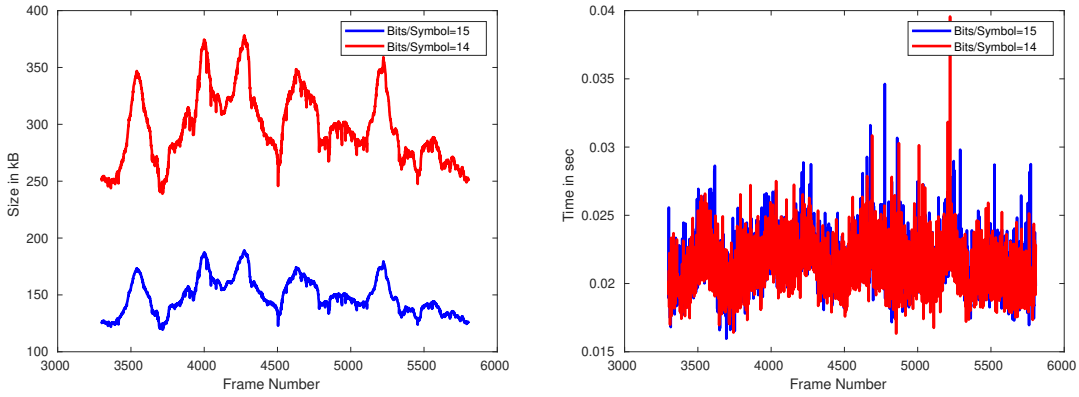


(**a**) (**b**)

Figure 3.4: Comparison between UNCC and *zlib* algorithms based on compression size and time.

A key difference between the UNCC compression technique and the *zlib* technique is that the UNCC technique allows the user to change the behavior of the compressed depth image by varying the bits per symbol (BPS) and the quantization step size (QSS). The BPS determines the number of bits in each symbol of the dictionary whereas QSS determines how close two neighboring entries in a dictionary can be. The QSS values are represented as the inverse of the distance threshold between depth entries and have the units of m^{-1}.

Fig. 3.5 shows the variations in the size of the compressed depth image and the total time taken to encode and decode a depth image for every frame in the dataset. We note that while the size of the encoded depth image changes significantly, the time taken to encode and decode the depth image does not change. Upon further investigation, we find that the majority of the time taken to perform the encoding operation is taken up by the bit-packing algorithms and the time taken to move data to and from memory. Hence, we notice no significant changes in time when both BPS and QSS parameters are varied. Table 3.4 summarizes the size and time when the parameters are varied.



(**a**) (**b**)

Figure 3.5: Plot showing the changes in the size of the compressed depth image and the total time takes to encode and decode a depth image when the bits per symbol are varied.

Table 3.4: Average size of the compressed image and total time while varying QSS and BPS

	Avg. Size of Compressed Image (kB)	**Total Time (sec)**
Quantization Step Size=1000 Bits/Symbol = 15	144.24	0.015
Quantization Step Size=700 Bits/Symbol = 15	446.81	0.02
Quantization Step Size=1000 Bits/Symbol = 14	297.87	0.021

3.5.2 Odometry Estimation

Odometry can be estimated using plane coefficients much faster than using depth and intensity measurements as DVO Odometry does. To the best of our knowledge, popular RGB-D SLAM systems perform very similarly in odometry estimation to DVO Odometry. Instead of matching depth for different frames, we match the algebraic representation of the surface itself. As discussed in Section 3.4.3, Plane Odometry, achieved by aligning two surface representations, avoids the effort of looking for corresponding measurements from surfaces at the corresponding locations. All of the points that are on the same surface would share the same plane coefficients, making the plane representation of the point cloud much more efficient. This is confirmed by our experiment results. We compare our Plane Odometry to our implementation of DVO Odometry in MATLAB and measure the computation time for odometry estimation. It is worth noting that DVO does an excellent job in architecture acceleration and is able to achieve real-time odometry computation. Our implementation of it is not metrically consistent with the wall-clock analysis of running their algorithm. Additionally, unlike the original DVO implementation with multi-scale calculation on the images, we only compute odometry at one single scale as the depth images and plane images have different sizes, which may require different scale numbers. Our experiments are conducted on the benchmark TUM RGB-D dataset. The results of our evaluation on 100 pairs of frames are presented in Fig. 3.6. It shows that our Plane Odometry is far more computationally efficient, running at about 12 times faster overall than DVO Odometry.

To take advantage of both image measurement and algebraic representation, as proposed in Section 3.4.3 we integrate our plane representation into DVO Odometry. In this Hybrid Odometry approach, by grouping depth data into 8×8 blocks and representing them with 4 coefficients (a, b, c, d), more computation savings can be achieved as visiting overall pixels can be avoided (although it actually introduces

more interpolation operations). The experiment results are also included in Fig. 3.6. Our Hybrid Odometry performs between DVO and Plane Odometry in terms of computational cost. It overall runs 3 times faster than DVO and 4 times slower than Plane Odometry.

The acceleration of odometry calculation compared to DVO odometry, a real-time implementation, suggests the real-time capability of our algorithms. Given that RGB-D image data is captured at a resolution of $N = 640 \times 480 \approx 307k$ and a frame rate of 30 images/second, these computational savings from using planar representation for odometry calculation may significantly affect the run-time of real-time image processing algorithms for this class of image sensors.

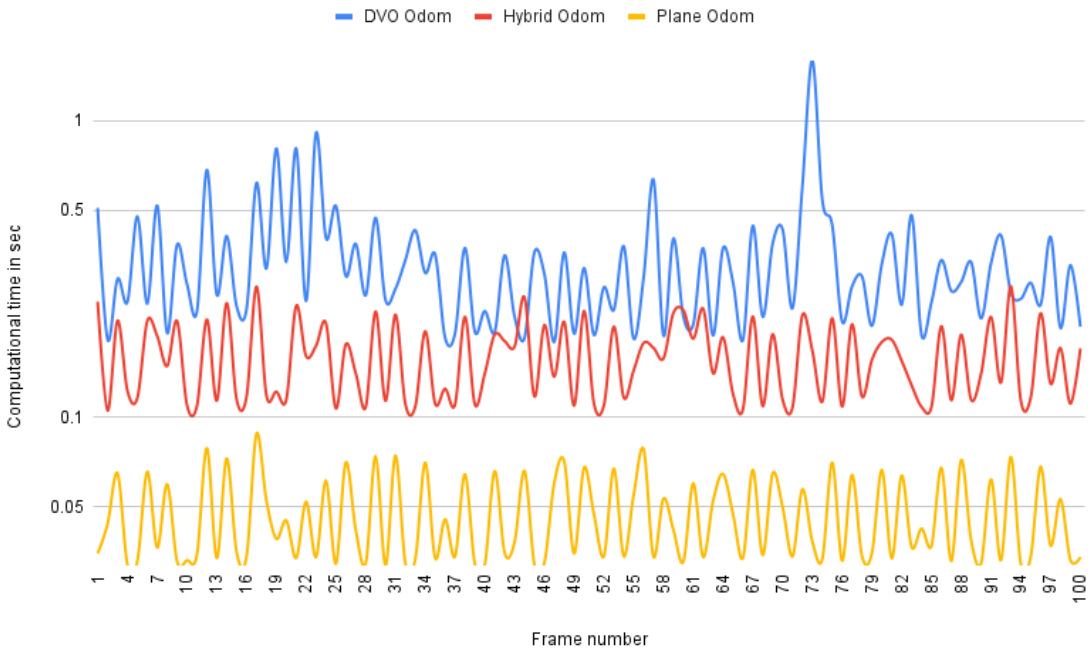


Figure 3.6: Average computational time of different odometry algorithms.

We also compare the accuracy of estimated odometry from the three aforementioned algorithms with ground truth which are shown in Table 3.5. While benefiting from handling much less data to achieve higher processing speed, both Hybrid Odometry and Plane Odometry suffer from larger errors and variances than DVO Odometry.

Table 3.5: Accuracy performance of different odometry algorithms (yaw, pitch, and roll are in degrees, and tx, ty, and tz are in meters).

	yaw (°)	**pitch** (°)	**roll** (°)	**tx** (m)	**ty** (m)	**tz** (m)
RMSE (DVO)	0.2741	03646	0.2042	0.0070	0.0028	0.0049
Std. Dev (DVO)	0.3585	0.4443	0.2969	0.0051	0.0042	0.0071
RMSE (Hybrid)	0.9417	1.7444	1.9219	0.0430	0.0441	0.0487
Std. Dev (Hybrid)	2.9798	5.9445	5.2683	0.1842	0.1830	0.1485
RMSE (Plane)	1.7984	1.9692	0.4452	0.0552	0.0489	0.1126
Std. Dev (Plane)	1.2027	1.0511	0.5727	0.0687	0.0312	0.0412

This provides an insight to balance the trade-off here based on the real application, for example, to achieve a good balance between speed and accuracy performance, in a multi-scale odometry calculation scenario, Plane Odometry can be applied in the higher level to initialize the estimator weights for lower-level DVO Odometry to leverage.

3.5.3 Semantic Segmentation

3.5.3.1 Network Training

Our modified version of RedNet architecture with a ResNet34 backbone is trained from random initialization on the preprocessed SUN RGB-D dataset in which the depth channel has been converted to a grid of plane coefficients. The SUN RGB-D dataset contains a vocabulary of 37 classes from indoor scenes. The optimizer used was Stochastic Gradient Descent with a learning rate of 0.002, a momentum of 0.9, and a weight decay rate of 0.0001. The training ran for 250 epochs on four NVIDIA 1080 GPUs.

3.5.3.2 Performance Evaluation

We evaluate our modified version of RedNet architecture on the test dataset from the preprocessed SUN RGB-D dataset. Overall, our network performs well on the modified SUN RGB-D dataset described above but compares unfavorably to the original implementation. This is partially attributable to the downsampling required to

calculate the plane coefficients, in this case shrinking the images by a factor of roughly four, and to the architecture not being structured to fully exploit the information in the plane coefficients. As a result, we achieved an overall pixel accuracy of 62.15% as opposed to 80.8% for the original RedNet and a mIoU of 0.272 as opposed to 0.468. The qualitative performance for three classes (floor, chair, and table) shown in Fig. 3.7a indicates that it fits for the purpose of generating semantic labels for mapping. Please note that both the ground truth (second row) and prediction (third row) show artifacts from repeated down/upsampling and lossy compression used solely for exporting the data to make this figure.

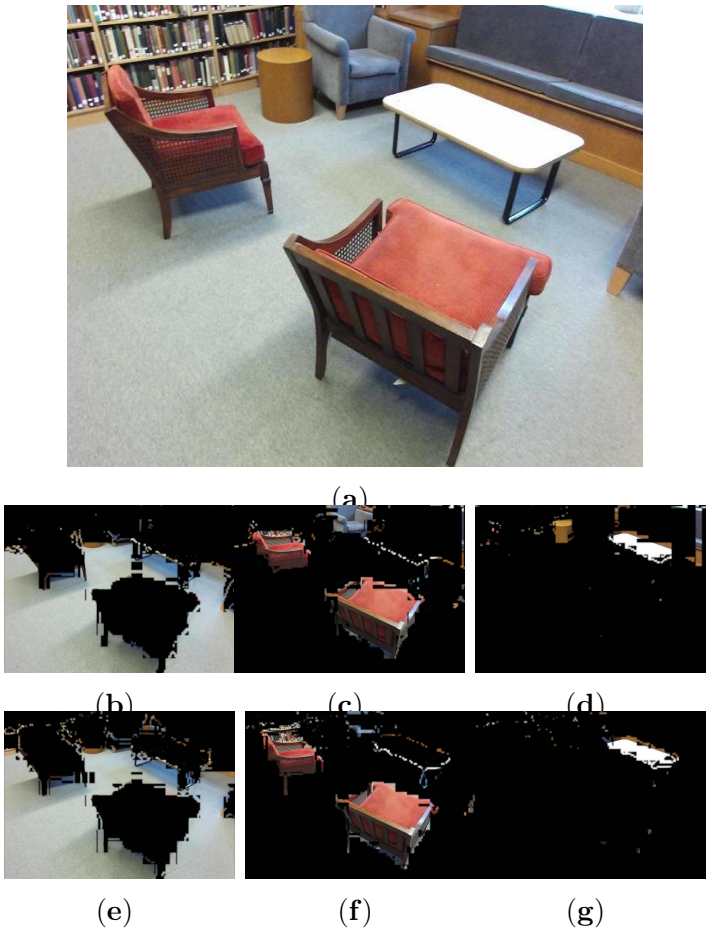


Figure 3.7: Qualitative results of an example image. (a) Example image. (b-d) Ground truth segmentation for three classes (floor, chair, and table). (c) Predicted segmentation for the three classes.

Those top-line statistics do not tell the whole story though. Table 3.6 shows the

Table 3.6: Mean IoU for classes in the SUN RGB-D dataset

Class	**mIoU**	**Class**	**mIoU**	**Class**	**mIoU**	**Class**	**mIoU**
Floor	0.766	Sofa	0.366	Whiteboard	0.235	Box	0.112
Ceiling	0.547	Window	0.358	Counter	0.208	Person	0.108
Wall	0.519	Mirror	0.340	Bookshelf	0.197	Night Stand	0.099
Chair	0.502	Cabinet	0.318	Lamp	0.187	Bag	0.068
Curtain	0.445	Door	0.296	Blinds	0.182	Shelves	0.058
Toilet	0.437	Fridge	0.294	Bookshelf	0.164	Curtain	0.005
Sink	0.421	Dresser	0.286	Desk	0.139		
Table	0.120	TV	0.279	Clothes	0.130		
Bed	0.412	Pillow	0.254	Towel	0.127		
Bathtub	0.411	Picture	0.250	Paper	0.121		

mean intersection over union (mIoU) for each of the SUN RGB-D classes. The mIoU values show that performance is much better on common classes that are well represented in the training data and classes that are well represented by large planes. Ceilings, walls, and floors have the highest mean pixel accuracy and mIoU, as is to be expected. Other classes such as toilets and sinks have unexpectedly high scores, presumably because they are isolated in view and protrude from walls and floors. For the top ten classes as ranked by mIoU, the accuracy for just those classes was calculated and listed in the third column. For the top three, the mean pixel accuracy is even higher than the overall accuracy of 81.3% reported for the original implementation of RedNet based on the larger ResNet-50 backbone and trained for longer on RGB-D data.

3.5.3.3 Semantic Map

We show an example of our label fusion results in Fig 3.8. By performing the warping we can align different keyframes if an overlap exists (second row). Label fusion will update the labels based on prediction confidence for all of the keyframe labeling results. From the last row of Fig 3.8, we can observe the consistency of labeling in different keyframes, for example, after performing fusion many pixels of the chair in the right image are modified and is consistent with the left image. Such consistency does not appear in the keyframes before the fusion (first row). Note

that this result is only for the purpose of displaying the process of label fusion. The labeling accuracy in terms of labels and associated confidence is dependent on both the network and the data we run experiments on, i.e. if the bagfile we use contains enough objects that the network can recognize after training. This is not the focus of the discussion.

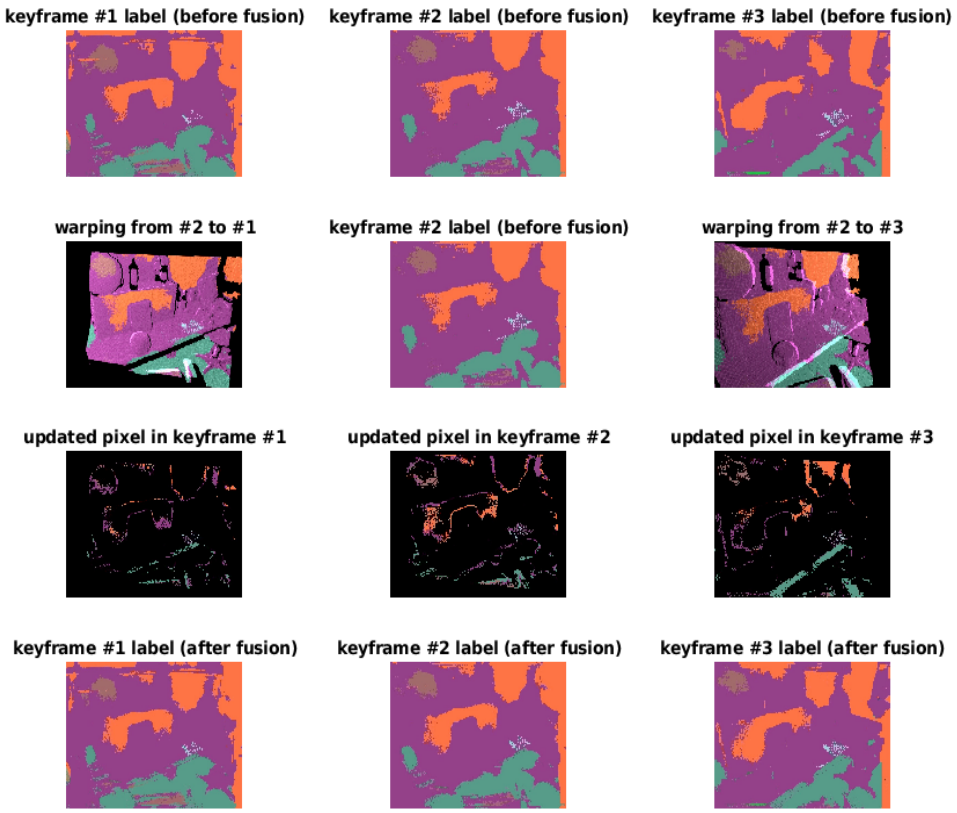


Figure 3.8: Label fusion: the first row shows three keyframe label images. The second row shows the reprojection of the image in the middle of the keyframe to the previous and next keyframe using the odometry measurement. The third row shows the pixels where the labels are updated by the fusion and the last row shows the results of label fusion. Labels across three keyframes are consistent after fusion.

We also present an example of our planar map with textures as well as with semantic labels in Fig 3.9. Note that these maps are represented by plane clouds which are a collection of smaller planar blocks instead of point clouds. The bagfile we use for

experiments does not contain sufficient objects that can be recognized by our trained network, thus the labels shown here are not necessarily perfectly reliable compared to the ground truth.

Figure 3.9: Global map with RGB appearance (left) and semantic map (right).

3.6 Conclusion

In this article, we propose a semantic SLAM system that uses planar surfaces to reduce the resources for communicating the complexity of the world. We propose two depth compression algorithms and integrate our plane-fitting algorithms on top of them. The plane fits can be computed efficiently and independently of the sensor's intrinsic camera parameters, and these planes can be used for many purposes such as fast odometry estimation. We also extend maps with semantic information predicted from sparse geometries by a CNN. Although the CNN architecture we adopt is not designated for the plane data and to be able to use it to train on plane data, we treat planar information as regular images, it still shows the potential to match semantic labels with planar models of the objects for efficient mapping. More research can be performed to address the aforementioned problems. With semantics and shape models available, our approach enables us to share among the robots very compressed knowledge of the world. In the future, we can explore more complicated geometry primitives for object representation. Estimating semantics and modeling rules at the same time is another interesting topic. The ultimate goal of this work is to

enable robots to achieve high-level tasking efficiently in distributed and collaborative settings.

REFERENCES

[1] F. Colas, S. Mahesh, F. Pomerleau, M. Liu, and R. Siegwart, "3d path planning and execution for search and rescue ground robots," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 722–727, IEEE, 2013.

[2] K. Pathak, A. Birk, S. Schwertfeger, I. Delchef, and S. Markov, "Fully autonomous operations of a jacobs rugbot in the robocup rescue robot league 2006," in *2007 IEEE International Workshop on Safety, Security and Rescue Robotics*, pp. 1–6, IEEE, 2007.

[3] S. Thrun, S. Thayer, W. Whittaker, C. Baker, W. Burgard, D. Ferguson, D. Hahnel, D. Montemerlo, A. Morris, Z. Omohundro, *et al.*, "Autonomous exploration and mapping of abandoned mines," *IEEE Robotics & Automation Magazine*, vol. 11, no. 4, pp. 79–91, 2004.

[4] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007.

[5] A. Birk, N. Vaskevicius, K. Pathak, S. Schwertfeger, J. Poppinga, and H. Buelow, "3-d perception and modeling," *IEEE robotics & automation magazine*, vol. 16, no. 4, pp. 53–60, 2009.

[6] S. Kohlbrecher, O. Von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable slam system with full 3d motion estimation," in *2011 IEEE international symposium on safety, security, and rescue robotics*, pp. 155–160, IEEE, 2011.

[7] J. Civera and S. H. Lee, "Rgb-d odometry and slam," in *RGB-D Image Analysis and Processing*, pp. 117–144, Springer, 2019.

[8] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," September 2014.

[9] R. Mur-Artal and J. D. Tardos, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-d cameras," *IEEE Transactions on Robotics*, vol. 33, pp. 1255–1262, oct 2017.

[10] C. Kerl, J. Sturm, and D. Cremers, "Dense visual slam for rgb-d cameras," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2100–2106, IEEE, 2013.

[11] C. Wang, J. Yuan, and L. Xie, "Non-iterative slam," in *2017 18th International Conference on Advanced Robotics (ICAR)*, pp. 83–90, IEEE, 2017.

[12] J. Engel, J. Stückler, and D. Cremers, "Large-scale direct slam with stereo cameras," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1935–1942, IEEE, 2015.

[13] R. Wang, M. Schworer, and D. Cremers, "Stereo dso: Large-scale direct sparse visual odometry with stereo cameras," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3903–3911, 2017.

[14] O. Kähler, V. A. Prisacariu, and D. W. Murray, "Real-time large-scale dense 3d reconstruction with loop closure," in *ECCV 2016*, pp. 500–516, 2016.

[15] T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J. J. Leonard, and J. McDonald, "Real-time large scale dense rgb-d slam with volumetric fusion," *International Journal of Robotics Research: Special Issue on Robot Vision*, vol. 34, pp. 598 – 626, April 2015.

[16] M. Rünz and L. Agapito, "Co-fusion: Real-time segmentation, tracking and fusion of multiple objects," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4471–4478, May 2017.

[17] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger, "Elasticfusion: Real-time dense SLAM and light source estimation," *The International Journal of Robotics Research*, vol. 35, no. 14, pp. 1697–1716, 2016.

[18] J. McCormac, A. Handa, A. Davison, and S. Leutenegger, "Semanticfusion: Dense 3d semantic mapping with convolutional neural networks," *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4628–4635, 2017.

[19] R. Dubé, A. Cramariuc, D. Dugas, H. Sommer, M. Dymczyk, J. Nieto, R. Siegwart, and C. Cadena, "Segmap: Segment-based mapping and localization using data-driven descriptors," *The International Journal of Robotics Research*, vol. 39, no. 2-3, pp. 339–355, 2020.

[20] S. Yang, Y. Huang, and S. Scherer, "Semantic 3d occupancy mapping through efficient high order crfs," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 590–597, IEEE, 2017.

[21] X. Li and R. Belaroussi, "Semi-dense 3d semantic mapping from monocular slam," *arXiv preprint arXiv:1611.04144*, 2016.

[22] Z. Xuan and F. David, "Real-time voxel based 3d semantic mapping with a hand held rgb-d camera," 2018.

[23] A. R. Willis, P. Ganesh, K. Volle, J. Zhang, and K. Brink, "Volumetric procedural models for shape representation," *Graphics and Visual Computing*, vol. 4, p. 200018, 2021.

[24] R. Liu and X. Zhang, "A review of methodologies for natural-language-facilitated human–robot cooperation," *International Journal of Advanced Robotic Systems*, vol. 16, no. 3, p. 1729881419851402, 2019.

[25] J. Zhang, A. R. Willis, and J. Godwin, "Compute-bound and low-bandwidth distributed 3d graph-slam," in *Unmanned Systems Technology XXII*, vol. 11425, p. 1142504, 2020.

[26] R. Finkel, "Quad trees: A data structure for retrieval on composite keys.," *Acta Inf.*, vol. 4, pp. 1–9, 03 1974.

[27] P.-Y. Lajoie, B. Ramtoula, Y. Chang, L. Carlone, and G. Beltrame, "DOOR-SLAM: Distributed, online, and outlier resilient SLAM for robotic teams," *arXiv preprint arXiv:1909.12198*, vol. 0, 2019.

[28] E. Montijano, R. Aragues, and C. Sagüés, "Distributed data association in robotic networks with cameras and limited communications," *IEEE Transactions on Robotics*, vol. 29, no. 6, pp. 1408–1423, 2013.

[29] E. Nettleton, S. Thrun, H. Durrant-Whyte, and S. Sukkarieh, "Decentralised SLAM with low-bandwidth communication for teams of vehicles," in *Field and Service Robotics*, pp. 179–188, Springer, 2003.

[30] D. Tardioli, E. Montijano, and A. R. Mosteo, "Visual data association in narrow-bandwidth networks," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2572–2577, IEEE, 2015.

[31] M. Hsiao, E. Westman, G. Zhang, and M. Kaess, "Keyframe-based dense planar SLAM," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5110–5117, IEEE, 2017.

[32] K. M. Brink, J. Zhang, A. R. Willis, R. E. Sherrill, and J. L. Godwin, "Maplets: An efficient approach for cooperative SLAM map building under communication and computation constraints," *IEEE/ION Position Location and Navigation Symposium*, April 2020.

[33] R. Dubé, A. Gawel, H. Sommer, J. Nieto, R. Siegwart, and C. Cadena, "An online multi-robot SLAM system for 3d lidars," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1004–1011, IEEE, 2017.

[34] T. Cieslewski and D. Scaramuzza, "Efficient decentralized visual place recognition using a distributed inverted index," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 640–647, 2017.

[35] T. Cieslewski and D. Scaramuzza, "Efficient decentralized visual place recognition from full-image descriptors," in *2017 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, pp. 78–82, IEEE, 2017.

[36] T. Bailey, M. Bryson, H. Mu, J. Vial, L. McCalman, and H. Durrant-Whyte, "Decentralised cooperative localisation for heterogeneous teams of mobile robots," in *2011 IEEE International Conference on Robotics and Automation*, pp. 2859–2865, IEEE, 2011.

[37] M. T. Lazaro, L. M. Paz, P. Pinies, J. A. Castellanos, and G. Grisetti, "Multi-robot SLAM using condensed measurements," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1069–1076, IEEE, 2013.

[38] J. Dong, E. Nelson, V. Indelman, N. Michael, and F. Dellaert, "Distributed real-time cooperative localization and mapping using an uncertainty-aware expectation maximization approach," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5807–5814, IEEE, 2015.

[39] S. Choudhary, L. Carlone, C. Nieto, J. Rogers, H. I. Christensen, and F. Dellaert, "Distributed mapping with privacy and communication constraints: Lightweight algorithms and object-based models," *The International Journal of Robotics Research*, vol. 36, no. 12, pp. 1286–1311, 2017.

[40] T. Cieslewski, S. Choudhary, and D. Scaramuzza, "Data-efficient decentralized visual SLAM," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2466–2473, IEEE, 2018.

[41] W. Wang, N. Jadhav, P. Vohs, N. Hughes, M. Mazumder, and S. Gil, "Active Rendezvous for Multi-Robot Pose Graph Optimization using Sensing over Wi-Fi," 2019.

[42] A. Trevor, J. Rogers, and H. Christensen, "Planar surface SLAM with 3D and 2D sensors," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 3041–3048, May 2012.

[43] V. Nguyen, A. Harati, A. Martinelli, N. Tomatis, and B. Sa, "Orthogonal SLAM: a step toward lightweight indoor autonomous navigation," in *In: Proceedings of the IEEE/RSJ Intenational Conference on Intelligent Robots and Systems, IROS*, 2006.

[44] R. F. Salas-Moreno, B. Glocken, P. H. J. Kelly, and A. J. Davison, "Dense planar SLAM," in *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 157–164, 2014.

[45] M. Kaess, "Simultaneous localization and mapping with infinite planes," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4605–4611, 2015.

[46] H.-Y. Shum, S. B. Kang, and S.-C. Chan, "Survey of image-based representations and compression techniques," *IEEE transactions on circuits and systems for video technology*, vol. 13, no. 11, pp. 1020–1037, 2003.

[47] R. Krishnamurthy, B.-B. Chai, H. Tao, and S. Sethuraman, "Compression and transmission of depth maps for image-based rendering," in *Proceedings 2001 International Conference on Image Processing (Cat. No. 01CH37205)*, vol. 3, pp. 828–831, IEEE, 2001.

[48] S. Mehrotra, Z. Zhang, Q. Cai, C. Zhang, and P. A. Chou, "Low-complexity, near-lossless coding of depth maps from kinect-like depth cameras," in *2011 IEEE 13th International Workshop on Multimedia Signal Processing*, pp. 1–6, IEEE, 2011.

[49] M. O. Wildeboer, T. Yendo, M. P. Tehrani, T. Fujii, and M. Tanimoto, "Color based depth up-sampling for depth compression," in *28th Picture Coding Symposium*, pp. 170–173, IEEE, 2010.

[50] S. Pratapa and D. Manocha, "Randm: Random access depth map compression using range-partitioning and global dictionary," in *Symposium on Interactive 3D Graphics and Games*, pp. 1–11, 2020.

[51] "zlib." https://zlib.net. Accessed: 2021-06-17.

[52] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017.

[53] J. Jiang, L. Zheng, F. Luo, and Z. Zhang, "Rednet: Residual encoder-decoder network for indoor rgb-d semantic segmentation," *arXiv preprint arXiv:1806.01054*, 2018.

[54] S. Song, S. P. Lichtenberg, and J. Xiao, "Sun rgb-d: A rgb-d scene understanding benchmark suite," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 567–576, 2015.

[55] A. Dai, M. Nießner, M. ZollhÃ¶fer, S. Izadi, and C. Theobalt, "BundleFusion," *ACM Transactions on Graphics*, vol. 36, pp. 1–18, jul 2017.

[56] T. Schops, T. Sattler, and M. Pollefeys, "Bad slam: Bundle adjusted direct rgb-d slam," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 134–144, 2019.

[57] X. Zhang, W. Wang, X. Qi, Z. Liao, and R. Wei, "Point-plane slam using supposed planes for indoor environments," *Sensors (Basel, Switzerland)*, vol. 19, 2019.

[58] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous robots*, vol. 4, no. 4, pp. 333–349, 1997.

[59] S. Thrun and M. Montemerlo, "The graph slam algorithm with applications to large-scale mapping of urban structures," *The International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 403–429, 2006.

[60] F. Dellaert and M. Kaess, "Square root sam: Simultaneous localization and mapping via square root information smoothing," *The International Journal of Robotics Research*, vol. 25, no. 12, pp. 1181–1203, 2006.

[61] E. Olson, J. Leonard, and S. Teller, "Fast iterative alignment of pose graphs with poor initial estimates," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 2262–2269, IEEE, 2006.

[62] R. M. Eustice, H. Singh, and J. J. Leonard, "Exactly sparse delayed-state filters for view-based slam," *IEEE Transactions on Robotics*, vol. 22, no. 6, pp. 1100–1114, 2006.

[63] K. Konolige and M. Agrawal, "Frameslam: From bundle adjustment to real-time visual mapping," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1066–1077, 2008.

[64] D. O. Wheeler, D. P. Koch, J. S. Jackson, T. W. McLain, and R. W. Beard, "Relative navigation: A keyframe-based approach for observable gps-degraded navigation," *IEEE Control Systems Magazine*, vol. 38, no. 4, pp. 30–48, 2018.

[65] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[66] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based SLAM," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, pp. 31–43, winter 2010.

[67] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G2o: A general framework for graph optimization," in *2011 IEEE International Conference on Robotics and Automation*, pp. 3607–3613, May 2011.

[68] F. Dellaert, "Factor graphs and gtsam: A hands-on introduction," tech. rep., Georgia Institute of Technology, 2012.

[69] J. Papadakis and A. R. Willis, "Real-time surface fitting to rgbd sensor data," in *SoutheastCon 2017*, pp. 1–7, March 2017.

[70] K. Khoshelham and S. O. Elberink, "Accuracy and resolution of kinect depth data for indoor mapping applications," *Sensors*, vol. 12, no. 2, pp. 1437–1454, 2012.

[71] J. A. Storer and T. G. Szymanski, "Data compression via textual substitution," *Journal of the ACM (JACM)*, vol. 29, no. 4, pp. 928–951, 1982.

[72] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[73] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu, and N. Sang, "Bisenet: Bilateral segmentation network for real-time semantic segmentation," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 325–341, 2018.

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

CHAPTER 4: FUSION OF SHAPE MODELS AND DEEP LEARNING

4.1 Introduction

The generation and understanding of three-dimensional (3D) geometries hold significant importance across diverse domains, from computer graphics to robotics and virtual reality. Recent advancements in deep learning (DL) and generative modeling have propelled research in the area of 3D shape generation. Notably, techniques such as Variational Autoencoders (VAEs) [1, 2, 3], 3D Generative Adversarial Networks (3D-GANs) [4, 5, 6], and 3D Stable Diffusion [7, 8, 9, 10] have shown promise to autonomously produce realistic and diverse 3D shapes. Shape grammars have also been demonstrated as a powerful approach for formal model generation by providing a rule-based framework for generating complex geometric structures and enforcing constraints within objects [11, 12, 13]. Each methodology has its advantages and limitations. This article seeks to provide a fusion of these two methodologies to achieve the best of both worlds for novel 3D shape synthesis.

Despite the progress achieved, challenges and limitations persist in deep learning 3D shape generation methodology. Figure 4.1 showcases several failure instances of these methodologies. VAEs, as evidenced in Figure 4.1(a), often struggle with capturing complex and high-dimensional distributions of 3D shapes. GANs often struggle with mode collapse in training, where the generator produces a limited diversity of shapes or collapses to a few modes, failing to capture the full distribution of the data, as shown in 4.1(b). Mode collapse can make GANs difficult to train and lead to the generation of unrealistic or repetitive shapes, limiting the variety and quality of the generated outputs. Figure 4.1(c) shows a car model generated by 3D stable fusion techniques which may struggle with preserving fine-grained details and local geometric

(**a**) (**b**) (**c**)

Figure 4.1: Deep learning methods face significant challenges in grasping the geometric and physical constraints inherent in 3D objects, resulting in shortcomings in the generated objects. (**a**) A VAE generated couch model only provides a rough approximation of the complex geometry of a couch [1]. (**b**) A 3D GANs generated table model lacks manifold geometry for the legs and fails to enforce self-similarity constraints, resulting in variations in shape and size among the four legs [4]. (**c**) A stable diffusion generated car model is smoothed on the object edges and fails to adhere to real-world constraints, as one of the front wheels occupies the spatial location intended for the car's front fender [9].

features, leading to information loss in complex shapes or smoothed shapes.

Shape grammars have been used commonly in computer graphics and design to describe the generation of complex shapes through a set of production rules [14, 15]. These rules define how basic shapes or components can be combined and manipulated to form more intricate structures. Shape grammars provide a systematic approach to generating shapes by specifying the relationships between various components and enforcing constraints to ensure the coherence and consistency of the generated designs.

Shape grammar rules can be implemented using the modeling language as the formal syntax and vocabulary [16, 17, 18, 19]. Users can define programs that describe objects as a semantic hierarchy of 3D shape elements where each element may be a semantic group of objects, e.g., a floor of a building, or an indivisible object, i.e., a brick within a building. Each indivisible object, e.g., a brick from a building, is modeled in terms of its geometry and appearance.

Shape grammars offer a means to encode the implicit generation rules and geometric constraints inherent in objects, which remains challenging for deep learning models to grasp. However, common objects typically have predictable generation rules which

are satisfied by all instances of these objects. For example, tables usually feature a top surface and multiple supporting legs connected to the top, and cars typically have four wheels on two sides that can roll. However, deep learning neural networks find it challenging to comprehend these constraints, hindering their ability to accurately generate such objects. Shape grammars provide a solution by implementing these rules to construct objects and allowing users to define the parameters of these rules.

Bridging deep learning techniques with shape grammars presents significant potential. Users can define shape programs and convey shape rules to artificial intelligence (AI) systems. By employing DL networks to learn the parameters of these rules rather than the rules themselves, AI models gain the ability to internalize such constraints. This capability leads to disentangled representations within the latent space, where each dimension corresponds to a meaningful attribute of the data. This promises enhanced controllability and interpretability of the generated shapes.

This article proposes a novel fusion of 3D shape representation using shape grammars and DL model estimation. Shapes are represented as a formal shape grammar using Procedural Shape Modeling Language (PSML) [19] which applies a sequence of rules to construct a 3D geometric model as a collection of 3D primitives. In contrast to competing approaches from the DL literature, the inclusion of dynamic parameterized formal shape models promises to allow DL applications to more accurately represent the structure of commonplace objects. In this article, we demonstrate several benefits of our approach that fuses shape models with DL estimation which are listed below:

- Shape estimates using can be guaranteed to satisfy complex geometric shape and physical constraints including self-symmetry, self-similarity, and free-standing stability properties.

- Shape estimates are guaranteed to satisfy important geometric model properties by providing water-tight, i.e., manifold, polygon models that require a small

number of triangle primitives to describe the basic object geometry.

- Shape estimates provide a highly compact parametric representation of objects allowing objects to be efficiently shared over communication links.

- User-provided shape programs allow human-in-the-loop control over DL estimates. Aspects of this control include specifying lists of candidate objects, the shape variations that each object can exhibit, and the level of detail or, equivalently, dimension of the latent representation of the shape. These aspects of our approach allow humans to more easily control the DL estimate outputs and also enable humans to more easily interpret DL estimate results which we collectively refer to as *"human-in-the-loop"* benefits.

- Users can control the complexity and diversity of DL-estimated shapes for each object and for each object component directly through the construction of the DL network.

- Object models can be used to synthesize training data for DL systems improving over current 3D model databases which use static 3D models and therefore lack geometric diversity. Object models can be combined to generate extremely large synthetic 2D/3D datasets having rich geometric diversity and including important annotations to support a wide variety of 3D and 2D DL applications.

- An example of the proposed DL fusion is provided that detects objects and their parametric representation given a PSML shape grammar is demonstrated. Key metrics for the model estimates are shown that demonstrate the benefits of this approach.

These contributions open the door to the integration of shape-grammar-based data generation methods with deep learning techniques for 3D object/scene understanding.

User-defined shape programs offer various benefits and advantages over competing approaches which are demonstrated by the results of this study.

4.2 Related Work

This study explores the advantages of shape grammar in data synthesis compared to other methods of data generation. Through experiments, the PSML-driven data generation approach shows significant potential for various computer vision applications. For these reasons, a review of the related literature to this article is divided into two parts:

- A overview of shape grammar and its applications.
- An examination of deep learning generative models of 3D shapes.

4.2.1 Shape Grammar

Shape grammar, proposed in the 1970s [20], provides a formal framework for generating and analyzing complex shapes and designs. As as a shape-based visual description grammar and a rule-based automated design grammar, shape grammar has been applied to many domains, including urban planning [12, 13], industrial design [14], and computer-aid design [15]. Over the years, advancements in shape grammar have led to the development of sophisticated methods for shape generation [21], analysis [22], and optimization [23], integrating computational techniques such as procedural modeling [19], parametric design [24], and machine learning [25].

There has been a considerable amount of work that investigates the use of shape grammars for vision tasks with a large number of articles being produced that focus on segmentation of architecture within images [26, 27] or segmentation of building facade images [28, 29, 30, 31]. However, these techniques were limited to 2D grammars since the labeled primitives produced by the used grammars were limited to 2D faces. Other work leverages shape grammar to model 3D indoor scenes from point

clouds [32]. Recently, researchers have been leveraging shape grammar to guide 3D shape semantic labeling [33] or scene graph generation [34]. This study systematically explores the benefits of the data generation method driven by the shape grammar and its potential in deep learning computer vision tasks and machine understanding where an AI system emulates the sense-making and decision-making ability of human beings.

4.2.2 Generative Models of 3D Shapes

Deep learning generative models have significantly advanced the field of 3D shape generation, with methodologies such as Variational Autoencoders (VAEs) [1, 2, 3], Generative Adversarial Networks (GANs) [4, 5, 6], and Stable Diffusion [7, 8, 9, 10] emerging as popular methodologies. VAEs encode input shapes into a latent space and reconstruct them via a decoder, but often struggle with capturing complex and high-dimensional distributions of 3D shapes. GANs leverage a generator-discriminator framework to produce realistic shapes but may suffer from mode collapse leading to the generation of unrealistic or repetitive shapes. Stable diffusion, a recent innovation, that offers improved training stability and control over generated samples, may struggle with capturing fine-grained details and preserving complex geometric properties present in real-world objects.

These methods train the AI systems to learn the geometric constraints such as self-similarity and physical constraints such as the free-standing stability of the 3D objects. This is extremely difficult for AI systems as their training data typically does not encode these rules or principles. It results that these methods tend to generate an approximation of instances of objects but not a geometric model that adheres to real-world constraints. The fundamental difference of this study is to encode into training data the organizations of objects in terms of their components and their relationships to other objects.

4.3 Methodology

The methodology of this article is organized into the following sections:

- An introduction to the Procedural Shape Modeling Language (PSML) that incorporates shape grammar programs as elements within the sequential programming code (Section 4.3.1).
- A discussion of the benefits offered by the PSML data generation approach (Section 4.3.2).
- A fused system of PSML and DL that takes point cloud as input and generates 3D shape estimates of objects (Section 4.3.3).
- An application of using the PSML-driven method to generate synthetic datasets (Section 4.3.4).

4.3.1 Procedural Shape Modeling Language (PSML)

PSML [19] is a programming language to generate 3D shapes procedurally based on shape grammar rules. It is similar in syntax and structure to Java but also incorporates shape grammar programs as elements within the sequential programming code. It provides programmers the ability to describe shapes in terms of their 3D elements where each element may be a semantic group of 3D objects, e.g., a brick wall, or an individual object, e.g., an individual brick. Modeling shapes in this manner facilitates the creation of models that more closely approximate the organization and structure of their real-world counterparts. As such, users may query these models for volumetric information such as the number, position, orientation, and volume of 3D elements. PSML grammar associates labels to both object-space, i.e. geometric objects, and void-space, i.e. the space that bounds objects. These labels may be used to facilitate analysis that requires knowledge of both types of information, e.g., furniture placement, accessibility, navigation of virtual spaces, path planning, etc.

Algorithm 1 The contents of PSML program: Table.psm

```
public class Table extends ShapeGrammar {
   public Table(Shape myShape, double l, double w, double h, double t, double offset_w, double
        offset_l) {
      double R = 0.73;
      double G = 0.55;
      double B = 0.39;
      rules {
         axiom::I("box", new double[]{l, h, w}) {table};
         table::split("y", new double[]{scope.s.y-t, t}) {bottom, top};
         top::appearance("diffuse", new double[]{R, G, B}){terminal};
         // round table top
         // top::I("cylinder", new double[]{w/2, t})appearance("diffuse", new double[]{R, G,
              B}){terminal};
         bottom::split("x", new double[]{offset_w, t, scope.s.x-2*t-2*offset_w, t,
              offset_w}){space, side, space, side, space};
         side::split("z", new double[]{offset_l, t, scope.s.z-2*t-2*offset_l,t, offset_l}){space,
              legMass, space, legMass, space};
         legMass::I("cylinder", new double[]{t/2, h-t}){leg};
         // square table legs
         // legMass::I("box", new double[]{2*leg_rad, 2*leg_rad, h-t}) R(Math.PI/2, 0, 0) {leg};
         leg::appearance("diffuse", new double[]{R, G, B}){terminal};
         space::void(){terminal};
      }
   }

   public static void main(String[] args) {
      Shape zShape = new Shape("root");
      Table s = new Table(zShape, 1, 1, 0.8, 0.05, 0.05, 0.05);
      s.showShapes("table");
   }
}
```

Algorithm 1 shows an example of a PSML program to generate a table object. The overall structure of a PSML program includes one *ShapeGrammar* declaration that contains one or more *method* declarations. Each method declaration must include at least one *rules* declaration. Typically, the body of each PSML grammar may include variable declarations and initialization as well as a collection of methods that are analogous to functions. Rule blocks must be defined within each method and each rules block contains a set of shape grammar production rules. Execution of a production rule causes the non-terminal symbol referred to as the predecessor to be replaced by the successor which may be one or more terminal or non-terminal symbols. Terminal symbols, indicated by the special string "terminal" in PSML algorithms, do not appear as predecessors in any production rule and are visible elements that exist in the final 3D model except terminals declared to be space. PSML is constrained to work entirely from closed geometries and associates semantic labels to non-terminals,

visible terminals, and empty spaces. In PSML, terminals are drawn from a pre-defined set of 3D shape primitives, for example, box, cylinder, sphere, and cone, and non-terminals are constructed from multiple terminals to represent complicated shapes. Shape grammars specified within the rule blocks use the passed shape, argument variables, and locally defined variables to generate an instance of the grammar shape.

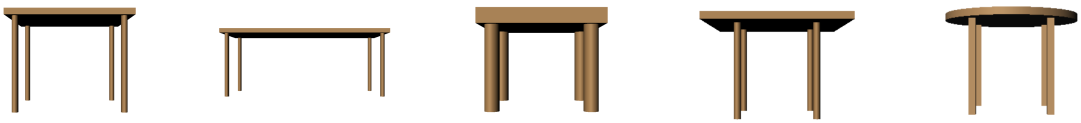
Algorithm 1 generates a "table" shown in Figure 4.2a. With the rule block of the table method, a box is first defined (line 7). The box is then split into two sections "*top*" and "*bottom*" (line 8) with the "*top*" being the surface of the table and the bottom the space where the table legs are. The algorithm then creates a space in the center of the "*bottom*" part and only keeps two faces on the side of the plane where the legs are (line 12). These two sides then respectfully get cut in the middle to finally create two table legs on each side (lines 13–14). The appearance of the table is colored brown (lines 9 and 17). The Table.psm generates a table object with a square top and four round legs. The commented lines (lines 10, 11, 15, and 16) offer the opportunity to generate a table object with different semantic structures, a round top, and square legs. In this example, the tabletop (line 19) and legs (line 17) are the "terminals" of the program and all other symbols are non-terminal symbols which are replaced by terminal or non-terminal symbols.

Figure 4.2 shows various realizations of the table object and demonstrates how the representation guarantees that target shapes satisfy the shape constraints over all possible parameter variations, e.g., there are always table legs connecting to a surface for these variations. More generally, PSML's syntax for detecting the size and position of the current volume allows the user to develop shapes that re-organize their components consistently over parametric variations, e.g., anisotropic scaling.

4.3.2 Benefits of PSML-driven Data Generation

Shapes represented using PSML offer significant benefits, including:

- Enforced geometric constraints and physical constraints.



(**a**) (**b**) (**c**) (**d**) (**e**)

Figure 4.2: Semantic variations of the table models generated by different PSML program parameters. (a) A visualization of the shape generated by Table.psm (Algorithm 1). (b) A variation of $l = 2$. (c) A variation of $t = 0.12$. (d) A variation of $offset\_l = offset\_w = 0.12$. (e) A variation with round top and square legs.

- Manifold polygon models.
- Compact parametric representation.
- Unlimited semantic variability.
- Human-in-the-loop control and interpretability of DL estimates.

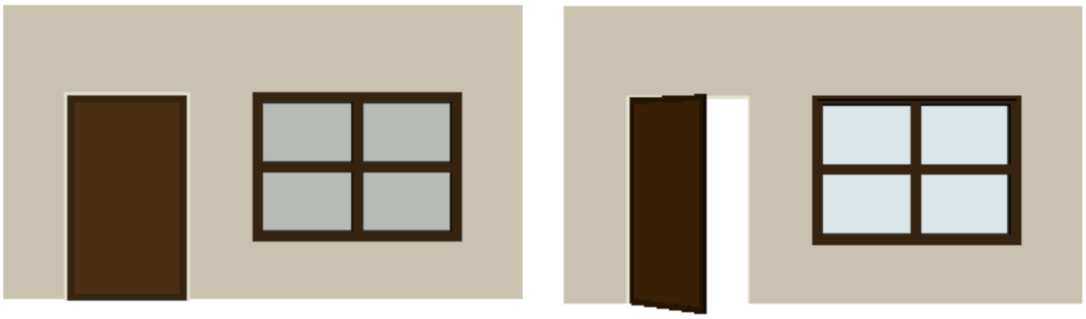
Each of these benefits will be discussed in the following subsections.

4.3.2.1 Geometric and Physical Constraints

Shapes generated using PSML are guaranteed to adhere to the geometric and physical constraints, including self-symmetry, self-similarity, and free-standing stability. By incorporating these constraints into the generation process, PSML ensures that the resultant shapes not only exhibit desired geometric properties but also possess structural integrity and functional coherence.

In the provided table example, the four legs are generated from the same shape primitive "cylinder" and the same parameters t and h (Algorithm 1 line 14). By employing this generation approach, PSML ensures that all legs exhibit precisely the same shape, thereby guaranteeing uniformity among them. This geometric constraint mirrors real-world manufacturing practices commonly employed in producing table objects, where consistency in leg design is important for structural stability.

PSML programming also allows components of objects to be constructed with appropriate relative positions. The relative position of the legs is delineated by the



(**a**) (**b**)

Figure 4.3: Shape grammar representation allows for the systematic generation of doors with realistic interactive behavior. (**a**) A closed door. (**b**) An open door.

common parameters, denoted as $offset\_l$ and $offset\_w$ (Algorithm 1 line 13). These geometric parameters define the spatial arrangement of the legs, ensuring consistency and symmetry in their positioning relative to each other and to the table top they support. The precise relative positions of the legs, together with the uniform shapes, promote balanced weight distribution and stability of the table, ensuring its suitability for applications such as real-world simulation.

Algorithm 2 The contents of PSML program: Door.psm

```
1  public Shape makeDoor(Shape myShape, double doorOpening, double frameWidth, double doorThickness) {
2      double angle = doorOpening * Math.PI / 2;
3      rules {
4          parent::T(-(myShape.s.x / 2) * Math.cos(angle) + myShape.s.x / 2, 0, - (myShape.s.x / 2) *
               Math.sin(angle)) R(0, - angle, 0)
5          I("box", new double[]{myShape.s.x, myShape.s.y, myShape.s.z}){door};
6          door::split("y", new double[]{frameWidth, myShape.s.y - 2 * frameWidth, frameWidth}){wood,
               mass1, wood};
7          mass1::split("x", new double[]{frameWidth, myShape.s.x - 2 * frameWidth, frameWidth}){wood,
               mass2, wood};
8          mass2::split("z", new double[]{(myShape.s.z - doorThickness) / 2, doorThickness,
               (myShape.s.z - doorThickness) / 2}){space, lightWood, space};
9          wood::appearance("diffuse", new double[]{0.2, 0.1, 0}){terminal};
10         lightWood::appearance("diffuse", new double[]{0.3, 0.15, 0}){terminal};
11         space::void(){j3d.terminal};
12     }
13     return myShape;
14 }
```

The constraints not only apply to individual objects but can also extend across different objects. In Figure 4.3, an example illustrates how a door generated using PSML can open and close within a wall. The PSML program for generating the door is outlined in Algorithm 2. The grammar rules (lines 2–5) dictate the mechanism

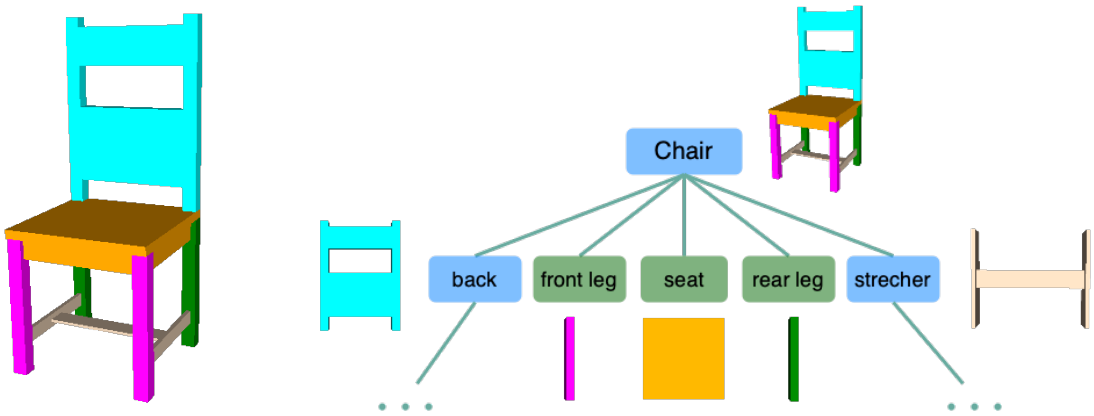
where the door can open or close by rotating along its side connected to the wall. By incorporating these constraints, the PSML program enables doors to exhibit realistic behavior when interacting with other objects, ensuring that the generated doors adhere to principles of real-world physics.

4.3.2.2 Manifold Polygon Models

Shapes generated by PSML are water-tight, i.e., manifold, models. This is attributed to PSML-generating objects from volumetric geometries, characterized by a pre-defined set of 3D closed-shape primitives such as boxes, cylinders, spheres, and cones. These primitives accurately describe the fundamental geometry of the object. This generation approach provides the ability to generate manifold geometries and represent objects as a semantic hierarchy of 3D shape elements. Other shape representations like point clouds and voxel meshes lack such properties. The components of these representations-points or voxels-operate independently, without constraints to enforce connectivity or the creation of a manifold geometry.

Figure 4.4 illustrates a chair instance generated from its hierarchical components. The chair is constructed from a set of components including a seat, front legs, rear legs, back, and stretchers. The shape derivation tree depicted in Figure 4.4b showcases the hierarchical composition. The blue nodes represent "non-terminal" objects whose child objects can further refine the shape of their parent objects by substituting the parent shape with one or more terminal or non-terminal shapes. The green nodes represent "terminal" objects, indicating that no further decomposition of this shape is available. These terminal objects, represented by boxes that are closed primitives, construct a chair object with a manifold polygon model. For simplicity, the shape derivation for the back and stretchers is omitted from the derivation tree.

This hierarchical representation of objects allows associating semantic labels to the components, which offers the opportunity for AI systems to understand objects at multiple levels of abstraction, from individual parts to complex assemblies.



(**a**) (**b**)

Figure 4.4: (**a**) An example of PSML constructing a chair hierarchically from its components. (**b**) Derivation tree of the chair. The components of the chair are colored. Shapes represented using PSML are constructed from their components and satisfy the relative constraints of the components.

4.3.2.3 Compact Parametric Representation

With pre-defined generation rules, shapes generated using PSML can be represented by a set of parameters that succinctly describe the geometry and appearance of objects. Parametric representations result in highly compact encoding of object information, minimizing the amount of data that needs to be transmitted or stored. This significantly reduces the dimensionality of the data compared to voxel-based or polygonal mesh representations.

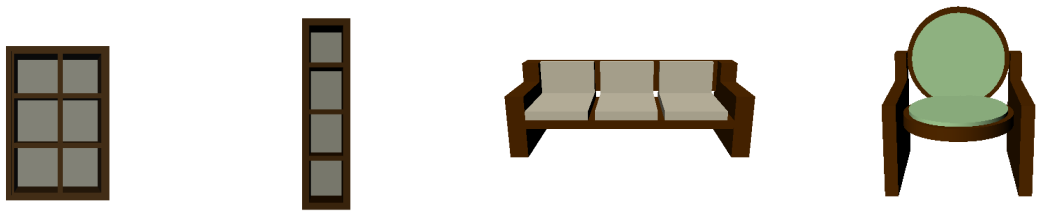
The parametric representation provided by PSML offers a lightweight yet powerful solution for encoding object information in resource-constrained environments, making it well-suited for applications like mobile robotics where efficient communication and collaboration are essential. In mobile robotic systems, where communication bandwidth is often limited, sharing detailed object representations like point clouds or meshes may be impractical due to their high data volume. However, if all robots share knowledge of the grammar of objects or scenes, they can simply communicate semantic labels and associated parameters to convey their understanding of the scene effectively. By transmitting only the semantic labels and relevant parameters, robots

can share information about the scene efficiently while minimizing data transmission overhead.

4.3.2.4 Unlimited Semantic Variability

Objects generated using PSML adhere to specific design rules and are defined by a set of parameters. Modifying the shape, size, or appearance of an object can be achieved by simply adjusting these parameters, rather than manipulating complex geometric data directly. This promises the unlimited semantic variability of object models which can be used to synthesize training data for DL systems improving over current 3D model databases which use static 3D models and therefore lack geometric diversity.

Figure 4.2 shows different variations of the table object generated by the Algorithm 1 using different PSML parameter values (Figure 4.2(b-e)) and rules (Figure 4.2(e)). The structure of the table object is controlled by the length l, the width w, the height h, the thickness of the tabletop t, and the position of the legs which are determined by the offset from the edges of the table, represented by $offset\_w$ and $offset\_l$ respectively. Figure 4.2(a-d) show the table variations generated by setting different values to these parameters. Figure 4.2(e) visualizes a table variation with round tabletop and square legs, which are opposite from other tables in Figure 4.2. More examples of other objects are shown in Figure 4.5 where variations of shelves and couches are generated by editing associated PSML programs, demonstrating the capability of generating unlimited semantic variations for DL systems. By applying different construction rules and/or passing different parameter values to the generation program of objects, indefinite variations of the object can be synthesized as training data for DL systems.



(**a**) (**b**) (**c**) (**d**)

Figure 4.5: (a) A shelf with 3 rows and 2 columns. (b) A single-column shelf. (c) A regular couch with 3 seats. (d) A couch with a round seat and back.

4.3.2.5 Human-In-The-Loop Control and Interpretability

User-provided shape programs allow human-in-the-loop control over DL estimates. Aspects of this control include specifying lists of candidate objects, the shape variations that each object can exhibit, and the level of detail or, equivalently, dimension of the latent representation of the shape. These aspects of our approach allow humans to more easily control the DL estimate outputs and also enable humans to more easily interpret DL estimate results which we collectively refer to as "human-in-the-loop" benefits. Users can control the complexity and diversity of DL-estimated shapes for each object and its components directly through the construction of the DL network.

Through parameterization, shapes generated using PSML allow to formalization of the DL estimate results into specific rules and constraints. Shape grammar rules incorporate parameters that define properties of generated elements, such as shape, size, orientation, and position. Parameters can also function within shape grammar rules to enforce constraints during generation. These parameters encode domain-specific knowledge of the shapes, for example, symmetry, alignment, and spatial relationships between components. When discrepancies arise between DL-estimated parameter values and ground truth, these errors serve as indicators of specific knowledge gaps within the model, such as its inability to learn rotational or positional relationships accurately. Analyzing the nature and frequency of these errors allows us to inform model improvement efforts, such as refining the network architecture or augment-

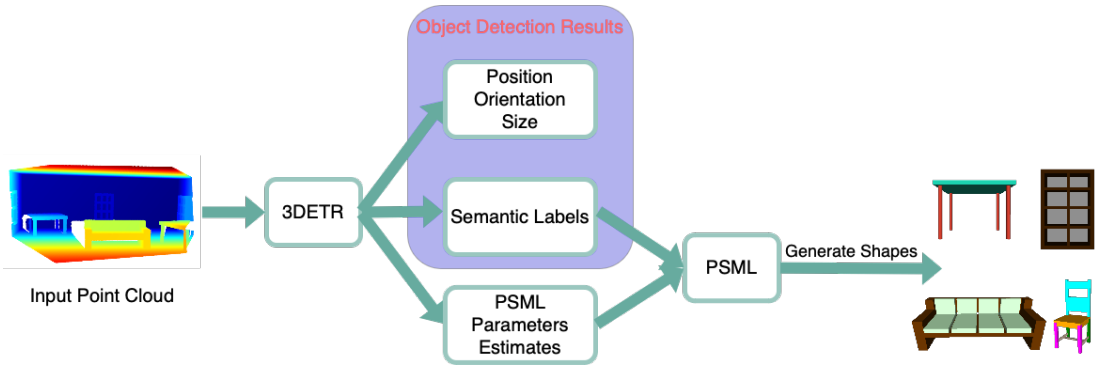


Figure 4.6: The fused system of PSML and DL for estimating 3D shapes.

ing the training data. The transparent attribution of errors to specific aspects of the input data enhances the interpretability of the model's results, enabling humans to understand and assess its predictions more effectively. This interpretability addresses a crucial challenge in making deep learning systems more comprehensible and trustworthy for real-world applications.

4.3.3 Fusion of PSML and Deep Learning

Figure 4.6 illustrates the fused system of PSML and DL for estimating 3D shapes. The proposed system takes the point cloud as input and outputs 3D shape estimates of objects. A DL network, 3DETR [35], was adapted and modified to perform 3D object detection and estimation of the PSML parameters. The estimated parameters, together with the semantic labels for determining the associated shape programs, are passed to the PSML to generate estimates of 3D shapes.

The 3DETR network [35] is an end-to-end Transformer-based object detection model for 3D point clouds. Unlike traditional convolutional neural networks (CNNs) which rely on spatial hierarchies to extract features from images, the 3DETR network leverages the self-attention mechanism of Transformers to capture both spatial and contextual information in an integrated manner. This enables the network to effectively process point cloud data, which lacks the grid-like structure present in images, while also facilitating global context understanding and precise localization of objects

within a 3D scene. The 3DETR network adapts an encoder-decoder architecture that produces a set of features. These features are fed into prediction Multi-Layer Perceptrons (MLPs) to predict bounding boxes. A 3D bounding box contains attributes including (a) location, (b) size, (c) orientation, and (d) the semantic class of the object.

Two modifications were implemented in the 3DETR network to facilitate the estimation of PSML parameters:

- A new multi-layer perceptron (MLP) was added to the existing architecture for PSML parameters estimation.
- The PSML parameters were encoded as an additional attribute of the 3D bounding boxes for prediction.

In this article, unless specified otherwise, the modified 3DETR network is denoted as 3DETR-P where P stands for PSML, while 3DETR refers to the original network. A vector of dimension 5 was chosen to encode the PSML parameters representing each object in the dataset. The dimensionality of this vector corresponds to the latent representation of the shape. By adjusting this size, DL networks are enforced to capture finer details of the object when increased, while reducing it provides more flexibility in the estimated solution space.

The $\mathcal{L}_1$ regression loss, i.e., the mean absolute error (MAE), was used as the loss function to measure the difference between the predicted values and the ground truth values. The equation for the PSML parameter loss is as follows:

$$\mathcal{L}_{\text{PSML}} = \frac{1}{n} \sum_{i=1}^{n} |p_i - \hat{p}_i| \tag{4.1}$$

where n is the number of PSML parameters, p_i is the ground truth value for the i-th paramter, and $\hat{p}_i$ is the network predicted value for the i-th parameter. This loss was

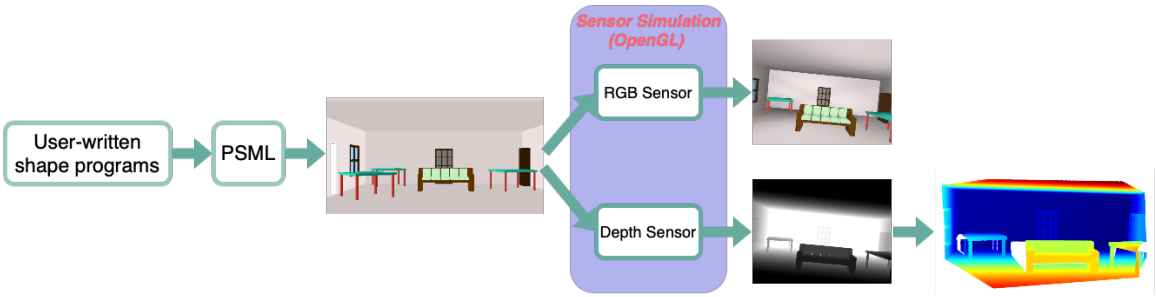


Figure 4.7: Different stages in the proposed data generation pipeline. The design of 3D models of scenes and objects happens in the PSML engine. These models are then passed to OpenGL where RGB and depth sensors are simulated for rendering RGB-D images and associated ground truth labels. Point clouds can be derived from depth data using the camera's intrinsic parameters.

added to the naive 3DETR loss with weight as the new final loss function to train the network. The final loss function is as follows:

$$\mathcal{L} = \mathcal{L}_{\text{3DETR}} + \lambda \mathcal{L}_{\text{PSML}} \tag{4.2}$$

where λ is the weight associated to the PSML loss and $\mathcal{L}_{\text{3DETR}}$ was defined in [35].

4.3.4 Data Synthesis for DL systems

Object models can be combined to generate extremely large synthetic 2D/3D datasets having rich geometric diversity and including important annotations to support a wide variety of 3D and 2D DL applications. This section describes a novel pipeline shown in Figure 4.7 to synthesize image data from user-written PSML programs. Physically realistic objects and/or scenes are first designed by shape grammar rules and created using PSML programs. These scenes are then passed to a rendering engine, for example, OpenGL to produce sensor data required by the users for their applications such as simulated RGB and/or depth image data, along with associated ground truth labels including 2D/3D bounding boxes, semantic segmentation labels, and PSML parameters.

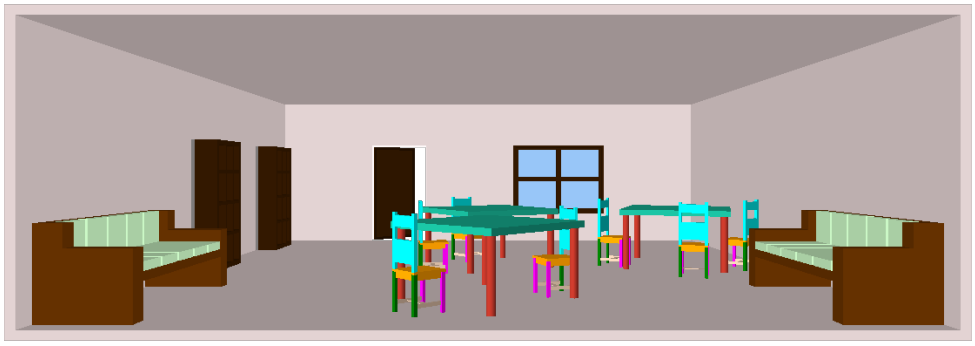


Figure 4.8: A room scene generated using PSML that combines multiple furniture objects.

4.3.4.1 Scene Data Generation

Figure 4.8 shows a room scene consisting of different objects including tables, chairs, couches, bookshelves, a door, and a window. Shape grammar and PSML are utilized to generate 3D designs of scenes and objects within them. Users first define shape grammar programs specifying the desired scene elements and their attributes. These programs are similar to Algorithm 1 and 2, outline the rules governing the structure, arrangement, and characteristics of the scene components. Subsequently, the PSML engine interprets and executes these shape grammar programs, generating physically realistic 3D designs of scenes. Through this process, the PSML engine determines the spatial relationships between objects, their shapes, sizes, orientations, and other relevant properties. Users can interactively adjust the parameters and rules within the shape grammar programs to refine the generated designs according to their preferences. This integration of shape grammar and PSML allows users to efficiently generate diverse and customizable 3D designs of scenes and objects.

4.3.4.2 Sensor Data Generation

OpenGL has been previously used by other researchers as a sensor simulator [36]. In this study, both sensor values and ground truth labels are generated through OpenGL by rendering the 3D models produced by PSML into 2D images. The process

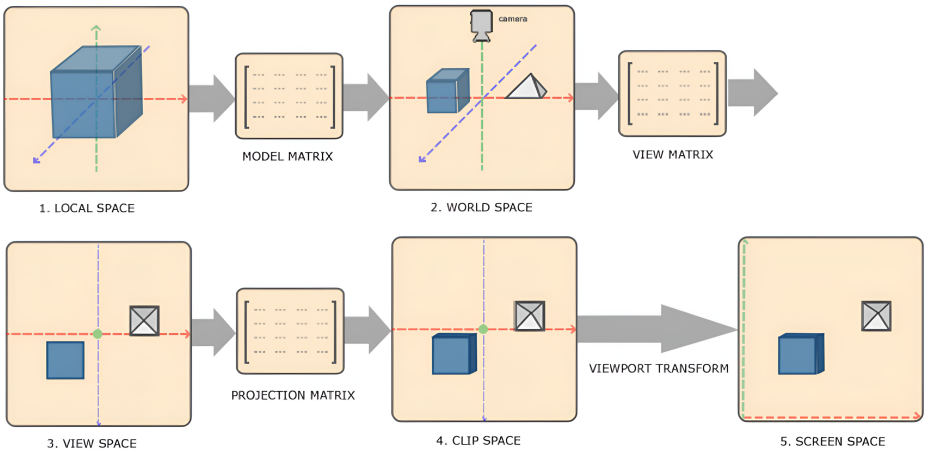


Figure 4.9: Vertex coordinate transformation from local space to screen space [37]. Object-relative vertex coordinates (local space) are converted to world coordinates (world space) using a model matrix $\mathbf{M}_{model}$, and then world coordinates are converted to view coordinates (view space) using a view matrix $\mathbf{M}_{view}$.

of converting 3D coordinates into 2D pixels is managed by the OpenGL graphics pipeline [37], which comprises two main parts: (1) projecting 3D surface coordinates (x, y, z) to their corresponding 2D locations (x, y) in the sensor image using the sensor projection model, and (2) assigning the value of these locations to the sensed values at the projected (x, y, z) location, representing the surface appearance for RGB images and the surface-to-sensor depth for depth images.

Figure 4.9 illustrates the OpenGL rendering pipeline and its internal transformations. In OpenGL, the transformation of local coordinates to screen (image) coordinates involves 4 steps: (1) Local-space coordinates, denoting the position of an object relative to its local origin, are transformed to world-space coordinates using a model matrix $\mathbf{M}_{model}$. These world-space coordinates represent the object's position relative to a broader world context and are referenced against a global origin shared by multiple objects within the scene. (2) The world coordinates are converted to view-space coordinates using a view matrix $\mathbf{M}_{view}$, aligning them with the perspective of the camera or viewer. This transformation ensures that each coordinate reflects the object's appearance from the viewpoint of the observer. (3) The view-space coordi-

nates are projected to clip-space coordinates using a projection matrix $\mathbf{M}_{projection}$, where they are processed to fit within the -1.0 and 1.0 range, determining which vertices will be visible on the screen. (4) The clip-space coordinates are transformed into screen-space coordinates through a process known as viewport transformation. The coordinates are mapped to the coordinate range defined by the viewport using the OpenGL function *glViewport*. Through this series of transformations, OpenGL accurately positions objects within the rendered scene, thereby generating the 2D OpenGL image. Step (1–3) can be represented in the following equation where $\mathbf{V}$ indicates vertex and $\mathbf{M}$ indicates transformation matrix:

$$\mathbf{V}_{clip} = \mathbf{M}_{projection} \cdot \mathbf{M}_{view} \cdot \mathbf{M}_{model} \cdot \mathbf{V}_{local} \tag{4.3}$$

The resulting screen coordinates are then forwarded to the rasterizer, where they are converted into fragments, each containing the necessary data for rendering a single pixel. The main purpose of the fragment shader is to calculate the final color of a pixel. Typically, the fragment shader contains data about the 3D scene, such as lighting, shadows, and light color, to determine the pixel's ultimate color.

OpenGL is also employed to simulate depth sensors through the utilization of the depth buffer. The depth buffer, created by the OpenGL windowing system, stores depth values as 16-bit floats within each fragment, representing the fragment's depth value. To mimic real depth sensors, noise consistent with actual sensors is introduced into these depth measurements, as documented in literature such as [38], which outlines observed accuracy for depth images from RGB-D sensors like the Microsoft Kinect sensor. This depth noise follows a Gaussian model, where depth variance increases quadratically with the sensor-to-surface depth. During rendering, OpenGL compares the depth values of each fragment with the current depth buffer. Fragments that are behind other fragments are discarded, while fragments that pass this depth test are rendered, and the depth buffer is updated with the new depth

values. This automated process, known as *depth testing*, is seamlessly handled by OpenGL.

4.3.4.3 Synthetic Dataset

The capability to generate 3D models and simulate sensors offers the flexibility to generate diverse datasets tailored to specific applications. In this study, a pin-hole camera model was used as the perspective model in OpenGL to simulate the sensors for an RGB-D image dataset creation. The poses of the sensors varied in different images. This was achieved by moving all objects in the scene in the reverse direction of camera movements, as OpenGL by itself is not aware of the concept of a camera [37]. Using the inverse camera model re-projection and the perfect depth map, it is also possible to calculate the 3D position of each surface in the scene. This integration of PSML and OpenGL for synthesizing data provides (1) RGB-D images, (2) the ground truth information of the object poses relative to the camera, (3) hierarchical decomposition of objects, and (4) parametric representation of objects, where (1), (3) and (4) benefit from PSML scene generation and (2) from the OpenGL sensor simulation.

This versatile data generation framework extends to diverse research goals, facilitating tasks such as city scene modeling with accurate labeling, object part segmentation with component-level ground truth labeling, and analysis of various object realizations to address data scarcity issues. Additionally, customization for different sensor types or views, such as fish-eye cameras or bird-view perspectives, and adjustments to illumination settings in OpenGL, further expand the framework's applicability across varied research domains.

4.4 Results

This section presents the results of three experiments. The results demonstrate the benefits offered by the PSML shape generation method and its fusion with deep

learning techniques.

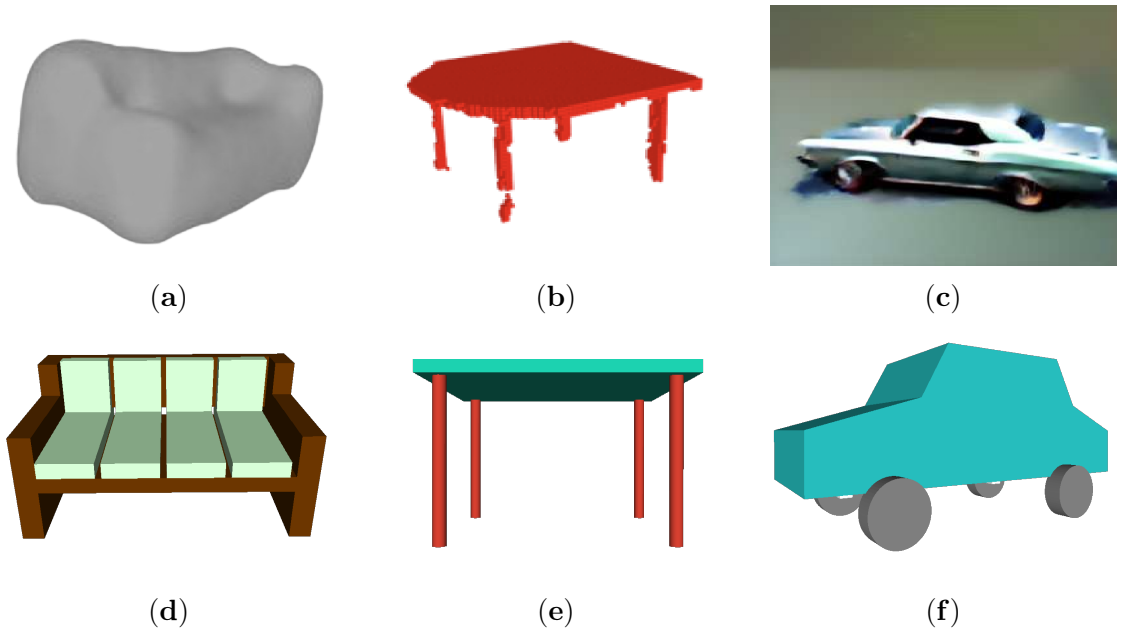
4.4.1 Comparison with Other Generative Methods

This experiment was conducted to demonstrate the advantages of 3D models generated using the PSML programs over the models generated by other competing methods. Specific cases of shapes generated by different methods were analyzed. From a wide array of possible algorithms, three algorithms representing VAE, GANs, and stable diffusion respectively, were evaluated against the PSML approach: (1) 3D Shape Variational Autoencoder (3DSVAE) [1], (2)3D Generative Adversarial Network (3DGANs) [4], and (3) Score Jacobian Chaining (SJC) [9]. While many algorithms are available in the literature, the selected algorithms provide a representative sampling of generative methods for 3D shapes.

Figure 4.10 illustrates the comparison between 3D shapes generated using PSML and using other approaches including VAE, GANs, and stable diffusion. The examples for comparison were sourced from their respective papers. The couch model generated by 3DSVAE (Figure 4.10(a)) lacks the structural characteristics of a couch object, offering only a rough approximation of its complex geometry. In contrast, the couch model generated using the PSML approach (Figure 4.10(d)) contains sufficient geometric features to represent a couch object. In the case of the table model generated by 3DGANs, the second leg from the left lacks manifold geometry, resulting in a discontinuous geometry and an unrealistic gap. Additionally, the legs lack self-similarity and self-symmetry in terms of size and length, which are typically present in real-world manufactured table objects. Conversely, the table model generated using PSML (Figure 4.10(e)) is a manifold polygon model and satisfies the geometric and physical constraints, attributed to its rule-based volumetric generation method. The car model generated by SJC (Figure 4.10(c)) lacks fine-grained details and fails to adhere to physical constraints, as one of the front wheels occupies the spatial location intended for the car's front. Its PSML-generated counterpart (Figure 4.10(f)),

however, presents a high-quality and physically realistic model.

Although the objects in Figure 4.10(d-f) may be rigid and visually simplistic, they do satisfy important common constraints for these commonplace objects. These constraints, for example, closed-shape geometry and free-standing capability, are necessary to be exhibited for objects to be classified into the correct category. The objects in Figure 4.10(a-c), while being visually sophisticated, would fail most realistic tests for symmetry and usability, for example, the couch cannot be sat on, the table would not stand, and the car wheels would not roll. The PSML approach ensures the generation of manifold 3D models that conform to important constraints. Building upon this technology to add more realistic details has the promise to achieve both visually compelling and reliable 3D geometry.



(**a**) (**b**) (**c**)

(**d**) (**e**) (**f**)

Figure 4.10: (**a**) A VAE-generated couch model [1]. (**b**) A 3D-GANs-generated table model [4]. (**c**) A stable-diffusion-generated car model [9]. (d-f) Models generated using PSML programs.

4.4.2 Comparison with Other Data Representations

This experiment was conducted to demonstrate the efficiency of the compact parametric shape representation offered by the PSML approach.

Figure 4.11 shows a table generated using Algorithm 1, and its polygonal mesh and point cloud representations. The shape grammar representation only requires 6 parameters (l, w, h, t, $offset\_w$, and $offset\_l$) to represent the geometry and 3 more parameters to describe the color appearance. In contrast, the polygonal mesh contains 674 vertices and 1328 triangular faces. The point cloud sampled from the mesh representation contains 5000 3D points. Assuming the data is represented using single-precision floating points (4 bytes), the total memory usage is 24,024 bytes for the polygonal mesh, 60,000 bytes for the point cloud representation, and only 36 bytes for the PSML parametric representation. The PSML representation requires to work with the associated program. Assuming each character in the Algorithm 1 is represented using 2 bytes, the program occupies 1,662 bytes, making the total memory necessitated for a PSML table object 1,698 bytes. Compared to the other two representations, this parametric representation reduces the data required to describe the geometry by $\sim$14 times compared to the polygonal mesh and $\sim$35 times to the point cloud.

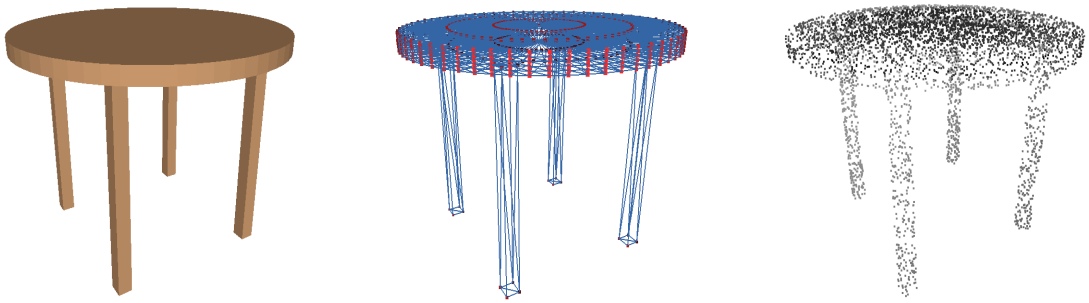
Table 4.1 illustrates the memory usage of three representations for various object instances. The memory usage for the point cloud representation was determined by sampling 2000 points per area unit of the mesh, while the PSML usage includes the memory required for the source code. The results indicate that PSML substantially reduces memory usage for most object instances, except for the chair, where polygon meshes achieved minimal usage. As the complexity of objects increases, such as in a room scene model containing various furniture pieces, the efficiency of PSML parametric representation becomes increasingly significant.

The results presented herein underscore the data efficiency offered by parametric representation in contrast to alternative methods. Through parameterization, the PSML approach retains considerable potential for achieving high data efficiency. This efficiency not only conserves memory but also streamlines the transmission and pro-

Table 4.1: Memory in byte required by different representations.

	Table (box)	Chair	Couch	Bookshelf	Window	Door	Room
PSML	1046	6410	3606	2350	2244	798	21328
Polygon Mesh	92496	3840	3360	15600	18960	1920	529344
Point Cloud	63360	42000	7063680	647760	786960	192240	14986080

cessing of object information, rendering it particularly advantageous for applications constrained by limited resources or bandwidth.



(**a**) (**b**) (**c**)

Figure 4.11: Shape grammar representation requires much less data to describe object geometry. (**a**) A table generated using Algorithm 1. Only 6 parameters are required to represent the geometry and 3 more parameters to describe the color appearance. (**b**) A polygonal mesh representation of the table with 674 vertices (blue) and 1328 points (red) each of which requires 3 parameters to represent the 3D coordinates. (**c**) A sampled point cloud from the mesh representation with 5000 3D points.

4.4.3 Deep Learning Integration

In this experiment, a synthetic dataset was generated, and the 3DETR-P network was trained on this dataset to detect 3D objects in the scene and estimate the associated PSML parameters.

4.4.3.1 Synthetic Dataset

An experiment was conducted to demonstrate that object models can be used to synthesize training data for DL systems, improving over current 3D model databases which use static 3D models and therefore lack geometric diversity.

A PSML program of the indoor room scene was written that involved other 6 PSML programs of common indoor furniture: table, chair, couch, bookshelf, window, and door. The proposed human-in-the-loop approach fixed various attributes of this shape-generation process and allowed other aspects to vary. Fixed aspects included the size of the room and some relative and physical constraints between objects including that (1) all of the objects are on the ground, (2) bookshelves are always against the wall, and (3) solid objects do not overlap with each other. The variations included occurrence, location, orientation, and structural characteristics of the furniture. This was achieved by controlling the PSML parameters for each object type. These parameters were set to follow uniform distributions and to ensure realism while adhering to relative constraints within and among objects. For example, the length and width of the table object were uniformly generated from 1 to 2 units, while the thickness ranged from 0.05 to 0.15 units. Similarly, the height of chair seats ranged from 0.5 to 0.8 units, reflecting real-world proportions where chairs typically sit lower than adjacent tables.

An RGB-D image dataset of the room scene was generated using the method in Section 4.3.4. The dataset was then utilized in a deep learning task for detecting 3D objects within the room, where the objective was to predict the 3D bounding boxes for each object based on the input point cloud. The point cloud data was derived from the depth data using the cameras' intrinsic parameters. Ground truth data was generated for each sample, comprising a semantic label, 3D bounding box (location, orientation, and size), and 5 PSML parameters. Specifically, for the bookshelf object, these PSML parameters included length, width, height (to define the object's 3D dimensions), number of horizontal panels, and vertical panels (to describe its structural characteristics). While the bookshelf necessitated all 5 parameters for PSML generation, it's important to note that not all objects require the same number of parameters. For instance, the door object in Algorithm 2 only requires 3 parameters

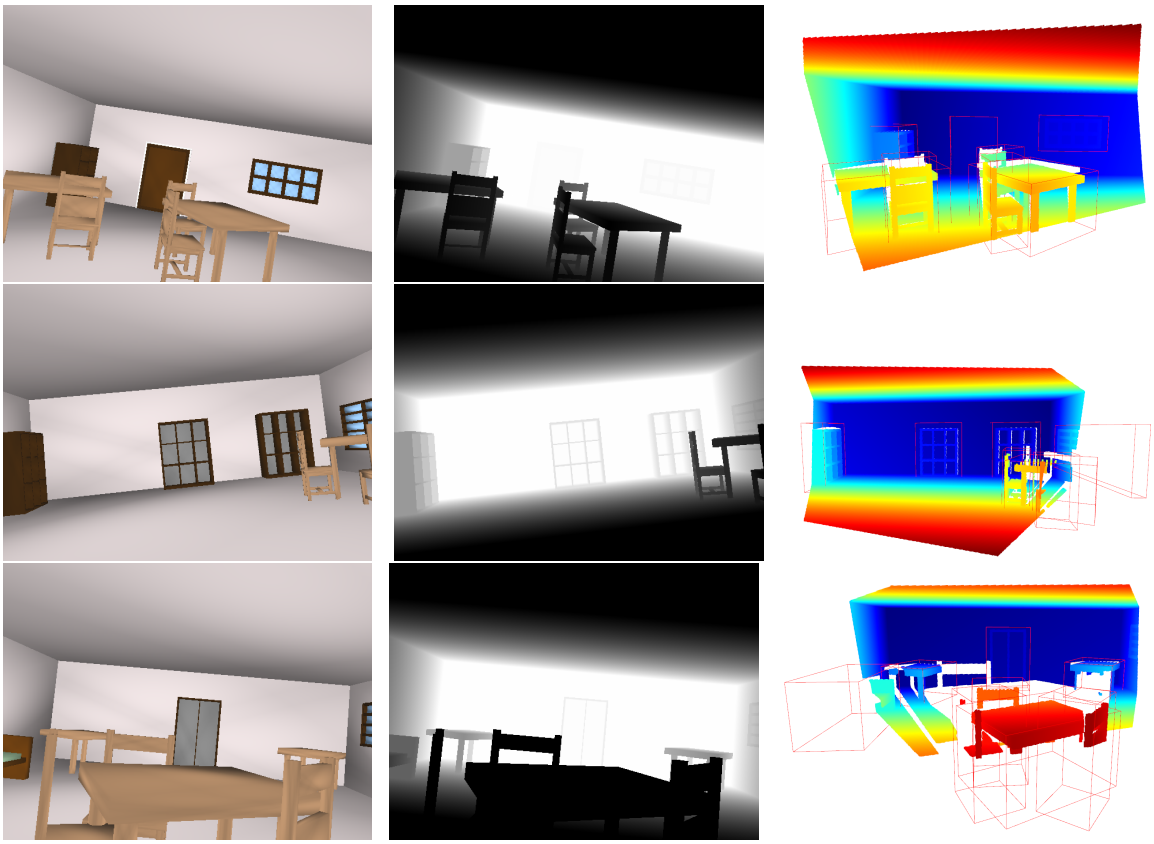


Figure 4.12: Examples in the synthetic dataset. Each row is one sample from the dataset. The images on each row from left to right are the RGB image, depth image, and point cloud with 3D bounding box ground truth shown in green.

as program arguments; in such cases, the extra 2 parameters are set to zero. The DL estimation of PSML parameters can be adjusted by increasing or decreasing the parameters for prediction. For example, limiting DL models to estimate only three parameters will result in less constraint within the DL solution space.

Figure 4.12 shows the RGB-D image pair and associated point cloud of 3 samples from the dataset containing 2000 samples. It can be seen that the occurrence of the objects, their shapes (length, width, height, etc.), positions, and orientations are different in the room space but still obey the physical constraints. Both RGB and depth images are rendered at a resolution of 640×480. The total processing time, including rendering and file writing, for each sample ranged from 1 to 2 seconds on an NVIDIA GeForce RTX 4090 GPU.

4.4.3.2 3D Object Detection

An experiment was conducted to demonstrate the capability of the proposed DL fusion in computer vision tasks, specifically detecting objects and their parametric representation. Key metrics for the model estimates are presented, highlighting the benefits of this approach.

The dataset generated in Section 4.4.3.1 was split into the train, validation, and test sets with 1200, 400, and 400 samples respectively (60%-20%-20%). The 3DETR-P network designed in Section 4.3.3 was trained to detect 3D objects in the scene and estimate the associated PSML parameters. The training was performed on an NVIDIA GeForce RTX 4090 GPU for 350 epochs with a batch size of 16. The weight for $\mathcal{L}_{\text{PSML}}$ loss in Equation 4.1 was set to 3. Other parameters were configured to be consistent with the [35].

Table 4.2 shows the testing results of the trained network. Following the practice in [35], the detection performance was reported on the test set using mean Average Precision (mAP) at two different IoU (Intersection of Union) thresholds of 0.25 and 0.5, denoted as AP_{25} and AP_{50}. The PSML parameters were evaluated by calculating the Mean Absolute Error (MAE) between the estimation and ground truth. The row corresponding to 3DETR-P in the table presents its performance on the room dataset created within this study. Overall, it succeeded in detecting objects within the scene, although its performance on door detection was comparatively lower. This discrepancy may be attributed to the fact that doors in the scene often (1) lack sufficient thickness to be distinctly separated from the wall they are embedded within and (2) lack sufficient depth variations within the object to provide more features for the network to learn the structure. The MAE_P row denotes the MAE of the PSML parameters, quantitatively showcasing the success of estimating the 3D shapes from the input point cloud.

Table 4.2 also includes the AP_{25} results of naive 3DETR on other datasets, re-

ported in [35]. The row corresponding to 3DETR-SUN reflects the 3DETR results from [35] on the SUN-RGBD dataset [39] and The 3DETR-SN row shows results on the ScanNetV2 dataset [40]. Although a direct comparison between the results in this article and theirs is not possible, it can be seen that 3DETR-P on the generated synthetic dataset achieved comparative detection performance than 3DETR on ScanNetV2 dataset for classes like chair, couch, and door, and outperformed 3DETR for other classes. The detection performance, together with the MAE results of the estimated PSML parameters, indicates the capability of the proposed PSML and DL fused system in detecting objects and their parametric representation.

Figure 4.13 visualizes three examples presenting the RGB image of the scene, the input point cloud, the ground truth and predicted 3D bounding boxes, and the 3D shapes estimated/reconstructed using the PSML parameters estimated by 3DETR-P. The appearance of the shapes was omitted as such information was not estimated by the network in this experiment. The reconstructed 3D shapes closely resemble those observed in the RGB images and the point cloud, thereby qualitatively demonstrating the success of 3D shape estimation from the input point cloud.

Table 4.2: Per-class performance for 3D object detection and shape estimation. MAE_P denotes the MAE of the PSML parameters. The 3DETR-P results were reported on the dataset generated in this article. The 3DETR-SUN row shows the 3DETR results from [35] on the SUN-RGBD dataset where "-" indicates such measurement is not available. The 3DETR-SN row shows the results of 3DETR on the ScanNetV2 dataset.

		Table	Chair	Couch	Bookshelf	Window	Door	Overall
3DETR-P	AP_{25}	98.90	87.60	99.30	98.95	93.89	56.84	89.25
	AP_{50}	89.64	61.76	95.16	93.96	62.51	13.16	69.36
	MAE_P	0.14	0.11	0.19	0.16	0.23	0.20	0.17
3DETR-SUN	AP_{25}	52.6	72.4	65.3	28.5	-	-	54.7
3DETR-SN	AP_{25}	67.6	90.9	89.8	56.4	39.6	52.4	66.1

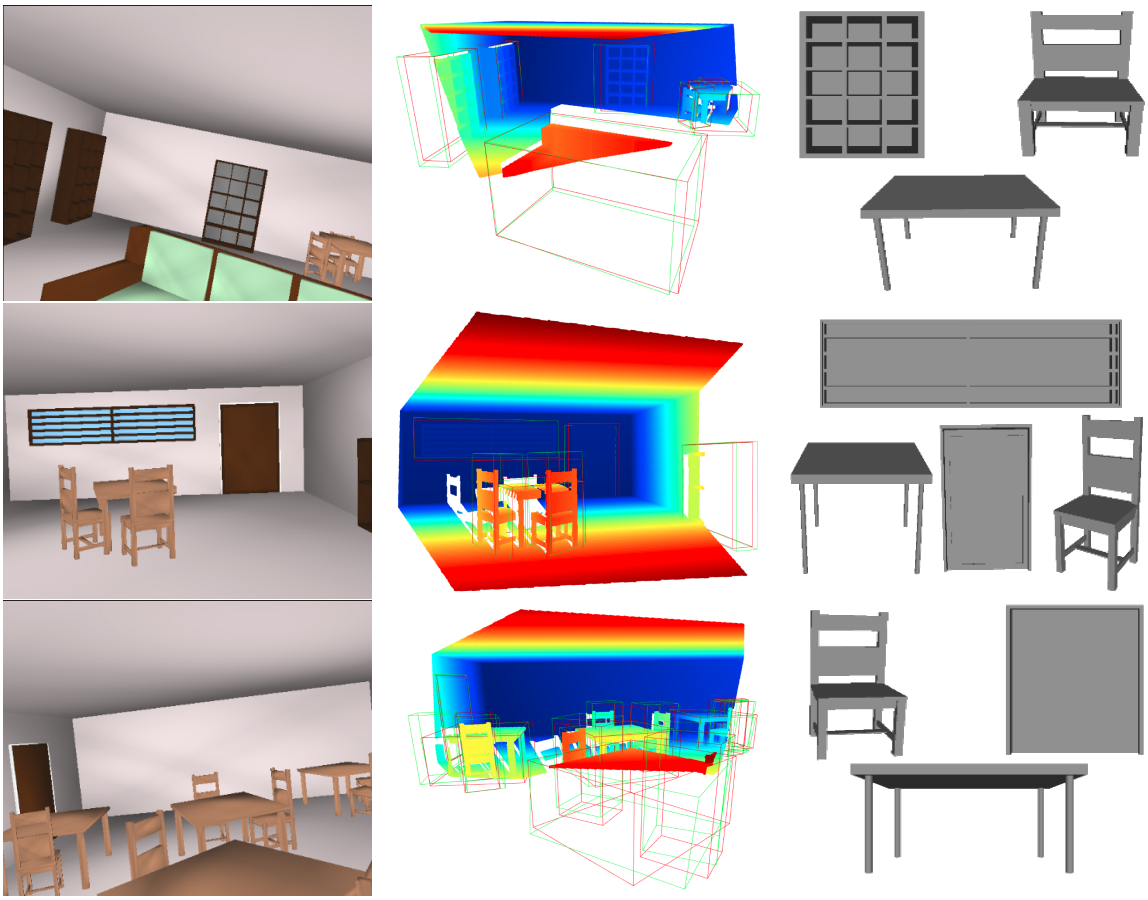


Figure 4.13: RGB image of scenes (left), input point cloud to the 3DETR-P with the ground truth and predicted bounding boxes in red and green respectively (middle), and some of the estimated 3D shapes reconstructed using the PSML parameters estimated by 3DETR-P and PSML programs.

4.5 Conclusions

This article introduced a novel fusion approach that combines a generative formal model for 3D shapes with deep learning (DL) methods to enhance the understanding of geometric structures and component relationships within objects. The proposed method leverages shape grammar programs written in Procedural Shape Modeling Language (PSML) to encode complex object descriptions. By allowing users to write PSML programs that enforce fundamental rules and encode object attributes, this fusion approach facilitates the generation of parametric representations for 3D shapes. One of the key strengths of the proposed approach is offering human-in-the-loop control over DL estimates, users can specify candidate objects, shape variations, and the level of detail, providing flexibility and control over the generated shapes. By enabling more accurate and controllable generation of 3D shapes, this fusion of generative modeling with DL enhances the interpretability of DL estimates and offers AI models a deeper understanding of object geometry and relationships, opening up new avenues for applications in computer graphics, robotics, and virtual reality.

REFERENCES

[1] E. A. Ajayi, K. M. Lim, S.-C. Chong, and C. P. Lee, "3d shape generation via variational autoencoder with signed distance function relativistic average generative adversarial network," *Applied Sciences*, vol. 13, no. 10, p. 5925, 2023.

[2] B. Dai and D. Wipf, "Diagnosing and enhancing vae models," *arXiv preprint arXiv:1903.05789*, 2019.

[3] A. R. Kosiorek, H. Strathmann, D. Zoran, P. Moreno, R. Schneider, S. Mokrá, and D. J. Rezende, "Nerf-vae: A geometry aware 3d scene generative model," in *International Conference on Machine Learning*, pp. 5742–5752, PMLR, 2021.

[4] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum, "Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling," *Advances in neural information processing systems*, vol. 29, 2016.

[5] A. Frühstück, N. Sarafianos, Y. Xu, P. Wonka, and T. Tung, "Vive3d: Viewpoint-independent video editing using 3d-aware gans," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4446–4455, 2023.

[6] E. R. Chan, C. Z. Lin, M. A. Chan, K. Nagano, B. Pan, S. De Mello, O. Gallo, L. J. Guibas, J. Tremblay, S. Khamis, *et al.*, "Efficient geometry-aware 3d generative adversarial networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 16123–16133, 2022.

[7] R. Liu, R. Wu, B. Van Hoorick, P. Tokmakov, S. Zakharov, and C. Vondrick, "Zero-1-to-3: Zero-shot one image to 3d object," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9298–9309, 2023.

[8] Stability AI, "3D couch generated using Zero123-XL models." https://stability.ai/news/stable-zero123-3d-generation. [accessed 07-Apr-2023].

[9] H. Wang, X. Du, J. Li, R. A. Yeh, and G. Shakhnarovich, "Score jacobian chaining: Lifting pretrained 2d diffusion models for 3d generation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12619–12629, 2023.

[10] J. Xu, X. Wang, W. Cheng, Y.-P. Cao, Y. Shan, X. Qie, and S. Gao, "Dream3d: Zero-shot text-to-3d synthesis using 3d shape prior and text-to-image diffusion models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 20908–20918, 2023.

[11] CityEngine, "http://www.esri.com/software/cityengine."

[12] L. Yang, J. Li, H.-T. Chang, Z. Zhao, H. Ma, and L. Zhou, "A generative urban space design method based on shape grammar and urban induction patterns," *Land*, vol. 12, no. 6, p. 1167, 2023.

[13] K. Zhang, N. Zhang, F. Quan, Y. Li, and S. Wang, "Digital form generation of heritages in historical district based on plan typology and shape grammar: case study on kulangsu islet," *Buildings*, vol. 13, no. 1, p. 229, 2023.

[14] M. Barros, J. P. Duarte, and B. Chaparro, "A grammar-based model for the mass customisation of chairs: modelling the optimisation part," *Nexus Network Journal*, vol. 17, pp. 875–898, 2015.

[15] I. Jowers, C. Earl, and G. Stiny, "Shapes, structures and shape grammar implementation," *Computer-Aided Design*, vol. 111, pp. 80–92, 2019.

[16] S. Havemann and D. Fellner, "Generative parametric design of gothic window tracery," in *Proceedings Shape Modeling Applications, 2004.*, pp. 350–353, IEEE, 2004.

[17] N. Goodman, V. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum, "Church: a language for generative models," *arXiv preprint arXiv:1206.3255*, 2012.

[18] S. Havemann, *Generative mesh modeling*. PhD thesis, Havemann, 2005.

[19] A. R. Willis, P. Ganesh, K. Volle, J. Zhang, and K. Brink, "Volumetric procedural models for shape representation," *Graphics and Visual Computing*, vol. 4, p. 200018, 2021.

[20] G. Stiny, "Introduction to shape and shape grammars," *Environment and planning B: planning and design*, vol. 7, no. 3, pp. 343–351, 1980.

[21] D. Ritchie, B. Mildenhall, N. D. Goodman, and P. Hanrahan, "Controlling procedural modeling programs with stochastically-ordered sequential monte carlo," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, pp. 1–11, 2015.

[22] H. Jiang, D.-M. Yan, X. Zhang, and P. Wonka, "Selection expressions for procedural modeling," *IEEE transactions on visualization and computer graphics*, vol. 26, no. 4, pp. 1775–1788, 2018.

[23] J. O. Talton, Y. Lou, S. Lesser, J. Duke, R. Mech, and V. Koltun, "Metropolis procedural modeling.," *ACM Trans. Graph.*, vol. 30, no. 2, pp. 11–1, 2011.

[24] M. P. Mata, S. Ahmed-Kristensen, and K. Shea, "Implementation of design rules for perception into a tool for three-dimensional shape generation using a shape grammar and a parametric model," *Journal of Mechanical Design*, vol. 141, no. 1, p. 011101, 2019.

[25] R. K. Jones, T. Barton, X. Xu, K. Wang, E. Jiang, P. Guerrero, N. J. Mitra, and D. Ritchie, "Shapeassembly: Learning to generate programs for 3d shape structure synthesis," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 6, pp. 1–20, 2020.

[26] P. Koutsourakis, L. Simon, L. Teboul, G. Tziritas, and N. Paragios, "Single view reconstruction using shape grammars for urban environments," in *IEEE International Conference on Computer Vision*, pp. 1–8, 2009.

[27] G. Kyriakaki, A. Doulamis, N. Doulamis, M. Ioannides, K. Makantasis, E. Protopapadakis, A. Hadjiprocopis, K. Wenzel, D. Fritsch, M. Klein, *et al.*, "4d reconstruction of tangible cultural heritage objects from web-retrieved images," *International Journal of Heritage in the Digital Era*, vol. 3, no. 2, pp. 431–451, 2014.

[28] B. Hohmann, U. Krispel, S. Havemann, and D. Fellner, "Cityfit: High-quality urban reconstructions by fitting shape grammers to images and derived textured point clouds," in *ISPRS International Workshop*, pp. 1–8, 2009.

[29] P. Zhao, T. Fang, J. Xiao, H. Zhang, Q. Zhao, and L. Quan, "Rectilinear parsing of architecture in urban environment," in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, 2010.

[30] O. Teboul, I. Kokkinos, L. Simon, P. Koutsourakis, and N. Paragios, "Shape grammar parsing via reinforcement learning," in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2273–2280, 2011.

[31] O. Teboul, L. Simon, P. Koutsourakis, and N. Paragios, "Segmentation of building facades using procedural shape prior," in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, 2010.

[32] H. Tran, K. Khoshelham, A. Kealy, and L. Díaz-Vilariño, "Shape grammar approach to 3d modeling of indoor environments using point clouds," *Journal of Computing in Civil Engineering*, vol. 33, no. 1, p. 04018055, 2019.

[33] R. K. Jones, A. Habib, R. Hanocka, and D. Ritchie, "The neurally-guided shape parser: Grammar-based labeling of 3d shape regions with approximate inference," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11614–11623, 2022.

[34] T. Liu, S. Chaudhuri, V. G. Kim, Q. Huang, N. J. Mitra, and T. Funkhouser, "Creating consistent scene graphs using a probabilistic grammar," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 6, pp. 1–12, 2014.

[35] I. Misra, R. Girdhar, and A. Joulin, "An end-to-end transformer model for 3d object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 2906–2917, 2021.

[36] J. O. Woods and J. A. Christian, "Glidar: an opengl-based, real-time, and open source 3d sensor simulator for testing computer vision algorithms," *Journal of Imaging*, vol. 2, no. 1, p. 5, 2016.

[37] J. De Vries, "Learn opengl," *Licensed under CC BY*, vol. 4, 2015.

[38] K. Khoshelham and S. O. Elberink, "Accuracy and resolution of kinect depth data for indoor mapping applications," *Sensors*, vol. 12, no. 2, pp. 1437–1454, 2012.

[39] S. Song, S. P. Lichtenberg, and J. Xiao, "Sun rgb-d: A rgb-d scene understanding benchmark suite," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 567–576, 2015.

[40] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "Scannet: Richly-annotated 3d reconstructions of indoor scenes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5828–5839, 2017.

CHAPTER 5: CONCLUSION AND OUTLOOK

In conclusion, this dissertation has addressed critical challenges in the field of visual SLAM, which is essential for enabling autonomous robots to navigate and operate effectively in unstructured environments. Through a comprehensive exploration of three key aspectsâenvironment simulation and sensor data processing, resource efficiency, and 3D object representationânovel techniques and methodologies have been developed to advance the state-of-the-art in visual SLAM systems.

The first part of this dissertation focused on simulating realistic environments and enhancing sensor data processing for mapping applications. By developing a simulation framework tailored for various operational contexts, algorithms could be refined and evaluated more effectively. Additionally, the introduction of a photometric correction model for thermal sensors contributed to more robust SLAM systems capable of processing diverse sensor data.

In the second part, the dissertation introduced a novel approach to address resource constraints in SLAM systems by leveraging planar semantic maps. This low-bandwidth, computational-bounded SLAM solution offers promising implications for real-world deployment, particularly in scenarios with limited computational resources or bandwidth.

The third part proposed an advanced method for 3D shape generation, integrating deep learning systems with shape grammars, to enhance the representation of commonplace objects within SLAM frameworks. This fusion approach not only improves the accuracy of object representation but also enables closer collaboration between humans and robots through more interpretable and human-in-the-loop models.

The contributions made in this dissertation collectively advance the capabilities of

visual SLAM systems, enabling agents to perceive, navigate, and interact with spatial environments more effectively in the digital age. By generating and communicating compressed map information within resource constraints, these advancements pave the way for broader adoption and deployment of autonomous robots across various domains.

Looking ahead, further research avenues emerge to build upon the foundation laid in this dissertation. Continued exploration of advanced simulation techniques, refinement of resource-efficient SLAM algorithms, and deeper integration of deep learning with geometric models offer promising directions for future investigation. Additionally, extending the applicability of SLAM systems to new domains and addressing emerging challenges such as multi-agent collaboration and lifelong learning will be crucial for advancing the field of robotics in the years to come.

In summary, this dissertation contributes significantly to the ongoing evolution of visual SLAM technology, with implications spanning from fundamental research to real-world deployment. By addressing key challenges and presenting innovative solutions, it lays the groundwork for future advancements that will shape the future of autonomous robotics.