國立清華大學電機資訊學院資訊工程研究所

碩士論文

Department of Computer Science

College of Electrical Engineering and Computer Science

National Tsing Hua University

Master Thesis

使用 MAVLink 通訊協定之無人機於大範圍三維高斯潑濺場景擷取與飛行路徑規劃之最佳化

Optimizing Drone Trajectory Planning for Capturing Large-Scale 3D Gaussian Splatting Scenes via the MAVLink Protocol

112062583

吳承遠

Cheng-Yuan Wu

指導教授：徐正炘 博士

Advisor: Cheng-Hsin Hsu, Ph.D.

中華民國 114 年 6 月

6, 2025

中文摘要

使用無人機捕捉三維高斯潑濺物體時，如何從眾多候選視角中選擇合適的輸入視角，以確保高品質的新視角合成是一項重大挑戰。這一挑戰更受到無人機資源限制（如電池續航力與網路頻寬）的影響。在本論文中，我們利用不確定性量化方法來評估各候選視角對於三維高斯潑濺的貢獻，進而優化無人機的飛行軌跡規劃。 具體而言，我們提出一套最佳化且高效的演算法，能夠在飛行過程中即時計算無人機的飛行路徑。據我們所知，此方法在現有文獻中尚未出現。此外，我們開發了一個模擬器，能夠同時模擬真實物理效果、無人機飛行視覺化，以及通信協定，作為我們演算法的完整評估平台，提供貼近實際情境的測試與驗證。 透過大量實驗，我們的研究成果顯示，相較於既有方法，我們的系統能夠：（i）在飛行過程中逐步建構高品質的 3DGS 物體、（ii）最終生成品質更優異的 3DGS 模型、以及（iii）在選定的視角下，以較少的輸入影像完成上述目標。

Abstract

Capturing 3D Gaussian Splatting (3DGS) objects using drones presents a significant challenge in selecting suitable input views from candidate poses to ensure high-quality synthesized novel views. This challenge is compounded by the inherent limitations of drone resources, such as battery life and network bandwidth. In this thesis, we employ uncertainty to quantify the contributions of individual candidate poses so as to optimize the computed drone trajectories. More specifically, we introduce optimal and efficient algorithms to compute drone trajectories on the fly, which, to the best of our knowledge, has never been done in the literature. Furthermore, we developed a simulator that simultaneously incorporates realistic physics effects, drone flight visualization, and a communication message protocol. This simulator serves as a comprehensive evaluation platform for our proposed algorithm, enabling thorough testing and validation under conditions that closely resemble real-world drone operations. Our extensive experiments revealed that, compared to the previous studies, our solution can: (i) incrementally construct good-quality 3DGS objects while following the computed trajectories, (ii) deliver final 3DGS objects with superior quality, and (iii) achieve the above goals with fewer input views captured at selected poses.

致謝

我要衷心感謝指導教授徐正炘在整個研究過程中給予的寶貴指導和支持。 我也感謝孫元駿教導我三維高斯潑濺的相關技術並協助我修改 MMM'25 會議論文。 接著，我要感謝李承澤協助我一同改進本文中所使用的演算法。 最後，我要感謝所有實驗室同學在我兩年研究生生涯中給予的鼓勵和幫助。

Acknowledgments

I would like to express my sincere gratitude to Professor Cheng-Hsin Hsu for his invaluable guidance and support throughout the course of this research. My heartfelt thanks also go to Yuan-Chun Sun for his insightful explanations on 3D Gaussian Splatting and his help in refining my MMM'25 conference papers. I am equally grateful to Cheng-Tse Lee for his contributions in improving the algorithms presented in this work. Finally, I would like to thank all my labmates for their encouragement and assistance throughout my two-year journey as a graduate student.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, several 3D representations have been adopted in emerging applications, such as urban planning [34], smart agriculture [28], search and rescue [58], heritage preservation [48], defect detection [30], film production [9], and robotics navigation [35]. Popular 3D representations include 3D meshes [59], 3D point clouds [19, 27], Neural Radiance Fields (NeRFs) [17, 40], and 3D Gaussian Splatting (3DGS) [29]. Compared to 3D meshes and 3D point clouds, NeRF and 3DGS often lead to better visual quality of *synthesized novel views* from a few input views of a natural or computer-generated 3D object [22]. Here, synthesized novel views[1] refer to the new views generated from positions and orientations which differ from those of the input views. High-quality synthesized novel views enable 6 Degrees-of-Freedom (6-DoF) interactions for human beings, and can be further analyzed by vision-based analytics for knowledge extraction.

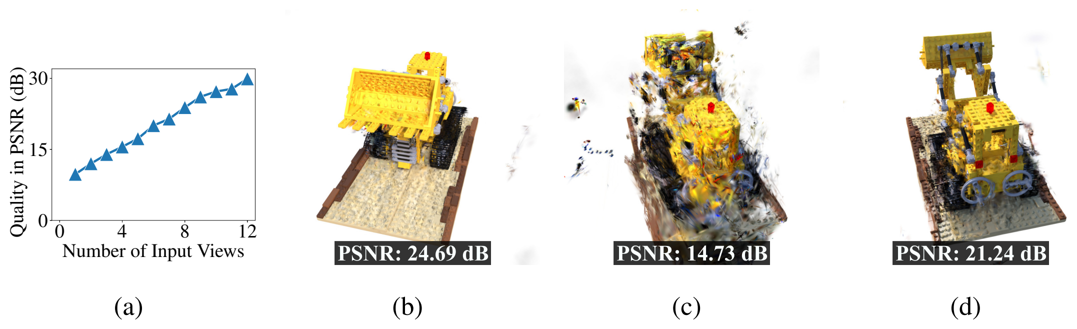


Figure 1.1: Pilot study results from: (a) different numbers of input views, (b), (c) sample synthesized views from clustered input views, and (d) a sample synthesized view from equally-spaced input views.

Compared to NeRF, 3DGS supports real-time synthesis, resulting in better responsiveness for interactive applications, and thus have attracted attention from both academia

[1]We use synthesized views for brevity in the rest of this paper.

and industry. While 3DGS demonstrates great potential, gathering input views for a larger 3DGS object [1, 5, 54], such as a building, landmark, bridge, cabin, vessel, or vehicle, is tedious, time-consuming, and error-prone. One way to capture input views from challenging positions and orientations is to employ a civilian drone with an RGB camera. However, it is crucial to navigate the drone to take a sufficient number of input views at different positions and orientations to train a 3DGS object that leads to high-quality synthesized views. In this paper, we collectively call the position and orientation of a drone to capture an image a pose, and refer to a series of poses as a trajectory.

To understand the importance of selecting input views, we conducted a pilot study using LEGO [40] in 3D meshes with the vanilla 3DGS implementation [29], which we describe as follows. We employed Unity [2] to render 24 equally-spaced views facing the center of the LEGO along the same latitude with a radius of four meters. We reserved the odd number views as testing views and present the 3DGS synthesized views in Fig. 1.1. First, we started from a single input view facing the bucket, and incrementally added an extra input view right next to the previous one. In total, we trained twelve 3DGS objects with increasingly more input views. We report the average quality of all synthesized views in Peak Signal-to-Noise Ratio (PSNR) [21] in Fig. 1.1(a), which demonstrates that more input views leads to better quality. With that said, each drone has limited resources, such as battery level and network bandwidth, and thus, we have to select the input views carefully. Next, considering six input views at different poses, we trained two 3DGS objects with (i) *clustered* input views close to the LEGO bucket and (ii) *equally-spaced* input views surrounding the LEGO. Figs. 1.1(b) and 1.1(c) give the synthesized views from the clustered input views with better (closer to the bucket) and worse (further away from the bucket) orientations, respectively. These figures show that a synthesized view away from the clustered input views suffers from catastrophic quality drops. Fig. 1.1(d) shows the synthesized view from the equally-spaced input views with the same pose as Fig. 1.1(c). This figure reveals a huge quality improvement (a 6.51 dB boost in PSNR), compared to the clustered input views, showing the importance of selecting input views.

Although our pilot study depicts the importance of input view selection, prior related works [7, 24, 43, 52, 56] only considered the Best View Selection (BVS) problem, which selects a set of input views from *already captured views*. Algorithms solving the BVS problem do not work well in online scenarios, where a 3DGS object is incrementally constructed on the fly when input views are captured by a drone. The evolving 3DGS object is used to select the next few poses for additional input views, forming the drone trajectory. We refer to this online version of the BVS problem as the Next Best View Selection (NBVS) problem, which, to the best of our knowledge, has never been studied on 3DGS objects, despite its NeRF variants having been recently investigated [25, 46, 63, 64].

In this paper, we study the NBVS problem to construct the drone trajectory for capturing each 3DGS object, which is challenging for two reasons. First, it is not easy to quantify the amount of information each potential, or *candidate* pose can bring to an existing 3DGS object. Second, it is non-trivial to systematically compute a drone trajectory to maximize the overall quality of synthesized views. To address the former challenge, we quantify the *uncertainty* of a candidate pose given the current 3DGS object. Since candidate poses with higher uncertainty levels can bring more additional information, they are more preferable during input view selection. To address the latter challenge, we mathematically formulated an optimization problem to compute the drone trajectory, which can be optimally solved by a Dynamic Programming (DP) algorithm. This DP algorithm, however, does not scale to bigger problem instances. Hence, we also propose an efficient heuristic algorithm for larger objects and real-time scenarios. Our extensive evaluation results show the strengths of the two proposed algorithms; they: (i) improved the visual quality of 3DGS objects by up to 5.90 dB in PSNR, (ii) cut the number of input views by up to 48.07%, and (iii) achieved the final synthesized view quality of the previous studies by up to 50+% time reduction.

1.1 Contributions

In this thesis, we makes the following contributions:

- We design and implement a drone simulator that closely resembles real-world drone operations. The simulator provides (i) realistic physics effects, including gravity, wind, and collision dynamics; (ii) photo-realistic rendering with dynamic lighting effects to simulate visual perception; and (iii) actual network packet exchanges through a full MAVLink protocol implementation. This platform enables comprehensive testing of drone behavior, perception, and communication under practical constraints.

- We propose two novel trajectory planning algorithms, DPC and AUM, that address the NBVS problem in 3DGS reconstruction. These algorithms utilize uncertainty estimation to quantify the information gain of individual candidate viewpoints, thereby enabling efficient and adaptive drone trajectory computation in real-time.

- We conduct a comprehensive experimental evaluation of DPC and AUM using our simulator and benchmark them against state-of-the-art offline reconstruction approaches. The results demonstrate that our methods not only produce higher-quality

3DGS models with fewer input views, but also significantly reduce computational overhead through online decision-making.

1.2 Organization

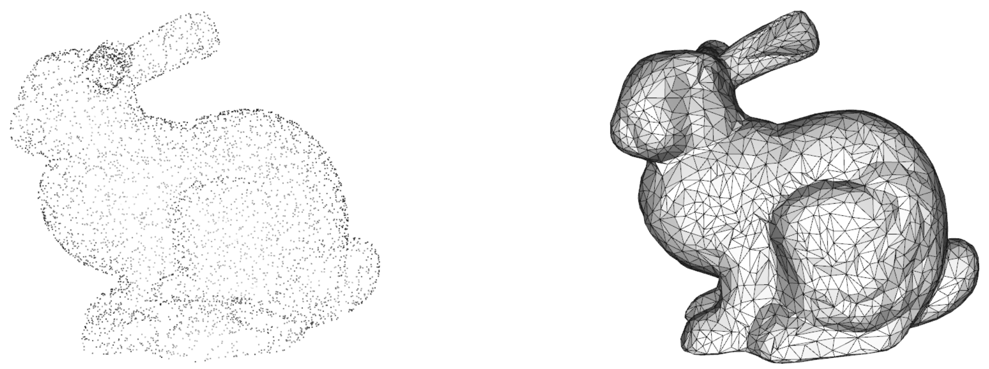
The remainder of this thesis is structured as follows: Chapter 2 provides an overview of commonly used 3D data representation formats and modern drone system. Chapter 3 reviews related work on the Next Best View Selection (NBVS) problem across different 3D representations, as well as existing drone simulation platforms. Chapter 4 describes the problem scenario considered in this thesis, focusing on drone-assisted 3D reconstruction. Chapter 5 introduces the formal problem formulations and details the proposed algorithms designed to address them. Chapter 6 presents the experimental setup and results used to evaluate the effectiveness of the proposed methods. Finally, Chapter 7 summarizes the key findings and outlines potential directions for future research.

Chapter 2

Background

In this chapter, we present an overview of commonly used 3D representation formats, followed by an introduction to the modern drone system.

2.1 3D Representations



(a) (b)

Figure 2.1: Examples of (a) point cloud and (b) mesh representations.

In this section, we review four commonly used forms of 3D representations: Point Clouds, Meshes, Neural Radiance Fields (NeRFs), and 3D Gaussian Splatting (3DGS). Each method presents a unique approach to modeling the geometry, appearance, and structure of three-dimensional environments.

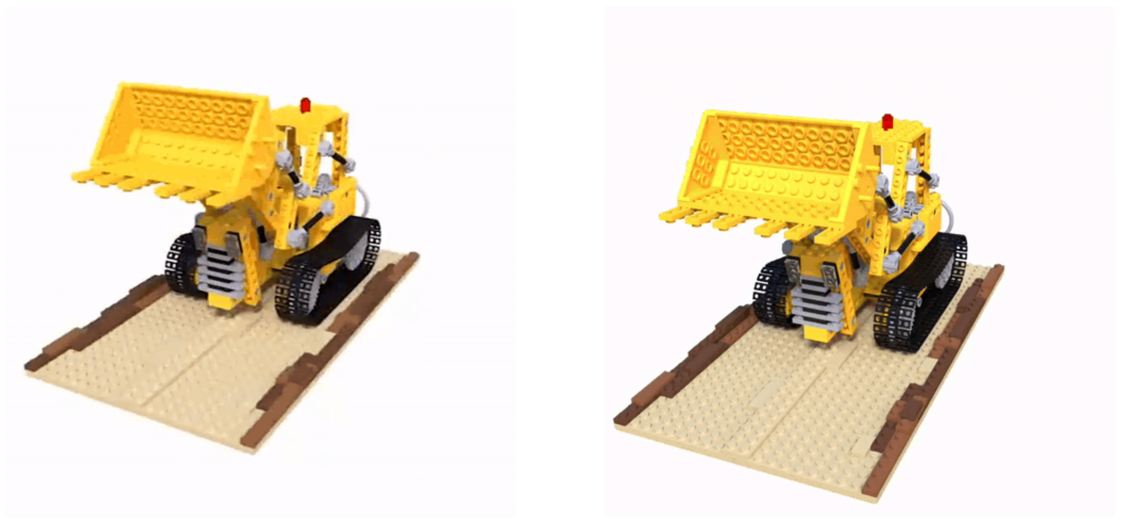
2.1.1 Point Cloud

Point clouds are one of the most widely adopted formats for representing 3D data. They consist of a set of discrete data points in a three-dimensional coordinate system, typically captured by depth sensors, LiDAR, or photogrammetry techniques. Each point contains spatial coordinates (x, y, z), and may optionally include additional attributes such as color, intensity, or surface normal vectors. As shown in Fig. 2.1(a), the point cloud representation models the object as discrete points, capturing its spatial distribution without explicit surface connectivity. Point clouds offer a flexible and efficient means of capturing the geometric structure of objects or environments. One of their primary advantages is their simplicity and ease of generation, especially from depth images or multi-view stereo techniques. Additionally, point clouds are lightweight compared to mesh or volumetric representations, making them suitable for real-time processing and transmission. However, point clouds also exhibit several limitations. They inherently lack topological connectivity, making it difficult to represent surfaces explicitly. This can lead to challenges in visualization, editing, and physical simulation. Furthermore, they are often sparse and noisy, particularly when generated from sensors with limited resolution or in environments with occlusions and poor lighting conditions. As a result, post-processing steps such as denoising, surface reconstruction, or meshing are often required to convert raw point clouds into more structured and usable 3D objects.

2.1.2 Mesh

Meshes extend the concept of point clouds by introducing connectivity between points. A mesh typically consists of vertices, edges, and faces, forming a polygonal structure, most commonly composed of triangles. As shown in Fig. 2.1(b), the mesh representation models the object's surface using connected triangular faces, clearly depicting its geometry. Meshes are highly effective for representing the surface geometry of 3D objects with precision and are widely used in computer graphics, simulation, and 3D modeling. The structured nature of meshes enables efficient rendering and physical simulation. Nevertheless, generating high-quality meshes from raw 3D data remains a non-trivial problem, especially when dealing with noisy or incomplete inputs. Despite these advantages, mesh comes with several challenges. The process of converting raw sensor data (e.g., point clouds or depth maps) into a mesh typically involves complex algorithms such as surface reconstruction, Poisson reconstruction, or Delaunay triangulation, which can be computationally intensive. Moreover, the resulting meshes may suffer from topological errors, such as holes, non-manifold edges, or poorly shaped triangles, especially in cases where the input data is sparse or noisy. Maintaining mesh consistency and integrity during incremental or real-time reconstruction remains a significant research challenge.

2.1.3 Neural Radiance Fields



(a) (b)

Figure 2.2: Examples of (a) NeRF and (b) 3DGS representations.

Neural Radiance Fields (NeRF) represent a novel paradigm in 3D scene reconstruction, leveraging deep learning to synthesize photorealistic images from novel viewpoints. Unlike traditional representations that rely on explicit geometry (e.g., point clouds or meshes), NeRF encodes a scene as a continuous volumetric function, typically implemented as a multilayer perceptron (MLP). To train a NeRF model, multiple images of a scene must be collected from different viewing angles. NeRF then predicts the corresponding color and volume density at each sampled 3D location. By integrating these values along rays, a photorealistic synthesized view of the scene can be generated. NeRF has demonstrated remarkable performance in novel view synthesis, achieving high-fidelity rendering of complex scenes with intricate lighting effects. However, NeRF's training process is computationally expensive and time-consuming, often requiring dozens of hours. Additionally, it lacks explicit geometry, making it difficult to perform tasks such as physical simulation or path planning in robotics.

2.1.4 3D Gaussian Splatting

3D Gaussian Splatting (3DGS) is a recently proposed approach that significantly improves both the efficiency and quality of 3D reconstruction compared to NeRF. Instead of using neural networks to represent color and density at every viewpoint in space, 3DGS represents a scene as a collection of anisotropic 3D Gaussians, each carrying parameters such as position, covariance, and radiance information. These Gaussians are rendered directly using rasterization-based techniques, enabling real-time rendering without the need for costly ray marching. Consequently, 3DGS offers an appealing trade-off between visual quality and computational efficiency. It retains the photorealistic appearance of NeRF while greatly reducing training and inference time, making it more practical for real-time or near-real-time applications. Moreover, it provides a more structured representation than NeRF, allowing for better integration with traditional graphics pipelines.

2.2 Drone system

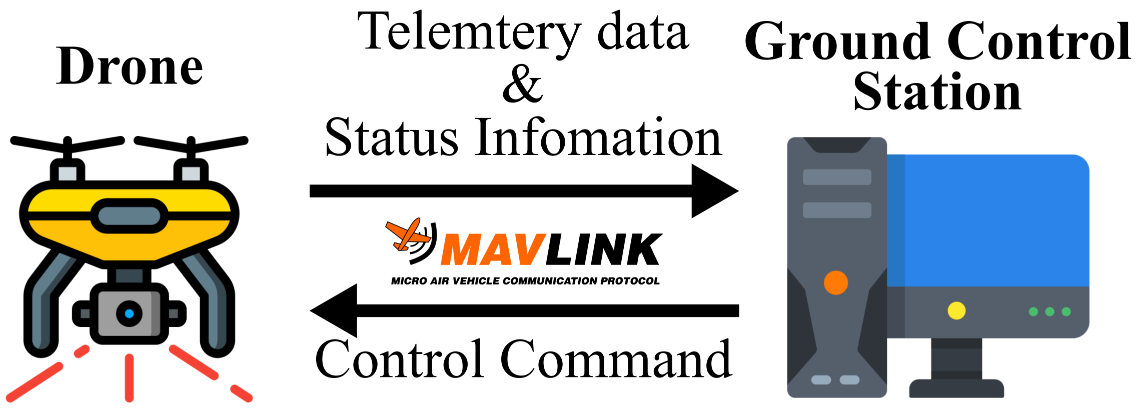


Figure 2.3: High-level architecture of the drone system

The high-level architecture of the drone system is illustrated in Fig. 2.3. Modern drone system is composed of multiple subsystems that work in tandem to achieve data collection, and real-time control, and autonomous flight. In this section, we briefly describe the three essential components that form the foundation of typical drone-based systems: Drone, Ground Control Station (GCS), and the communication protocol, typically implemented using MAVLink.

2.2.1 Drone

The drone, or unmanned aerial vehicle (UAV), serves as the aerial platform responsible for executing flight missions and collecting data. It is typically equipped with a flight controller, actuators, and sensors to collect images or videos. The common drone types are introduced as follows:

- Multirotor: These drones (e.g., quadcopters, hexacopters) are characterized by multiple rotors that provide vertical lift and enable precise hovering, vertical takeoff and landing (VTOL), and agile maneuverability. Their compact size, mechanical simplicity, and ease of control make them ideal for applications such as aerial photography, inspection, mapping, and indoor navigation. However, they generally suffer from limited flight duration and range due to high energy consumption required for sustained lift.

- Fixed-wing: Unlike multirotors, fixed-wing drones generate lift via aerodynamic surfaces and are propelled forward by one or more motors. They are capable of covering larger distances at higher speeds with greater energy efficiency, making them well-suited for long-range surveillance, agricultural monitoring, and environmental surveys. On the downside, fixed-wing drones typically require runways or catapults for takeoff and cannot hover.
- Hybrid (e.g., VTOL fixed-wing platforms): These platforms aim to combine the benefits of both categories, featuring vertical takeoff capabilities alongside efficient fixed-wing cruising. Hybrid drones are increasingly used in complex missions that demand both hovering precision and long endurance.

To enable autonomous operation, an autopilot system ,such as Pixhawk [38] or ArduPilot [4], is essential for drones. An autopilot is an integrated control unit that enables the drone to execute complex flight maneuvers automatically by processing onboard sensor data and following pre-programmed flight paths. Functioning as an embedded computer, the autopilot runs real-time control software that governs the drone's flight dynamics and navigation. These systems typically integrate a variety of sensors, including Global Positioning System (GPS) modules, Inertial Measurement Units (IMUs), and communication interfaces, allowing for precise localization, stability control, and decision-making without requiring continuous human input.

2.2.2 Ground Control Station

The Ground Control Station (GCS) serves as the central interface between the human operator and drone. It provides the tools necessary for mission planning, real-time monitoring, telemetry analysis, and manual override control. A GCS can range from lightweight laptop-based systems to dedicated ground terminals with multiple displays and control peripherals. Functionally, the GCS communicates with the drone's autopilot system via telemetry links, commonly using communication protocols such as MAVLink. Through this link, it receives real-time flight data, including position, attitude, battery status, and sensor readings, while also transmitting control commands or mission updates to the drone. Most modern GCS platforms support graphical user interfaces (GUIs) that allow

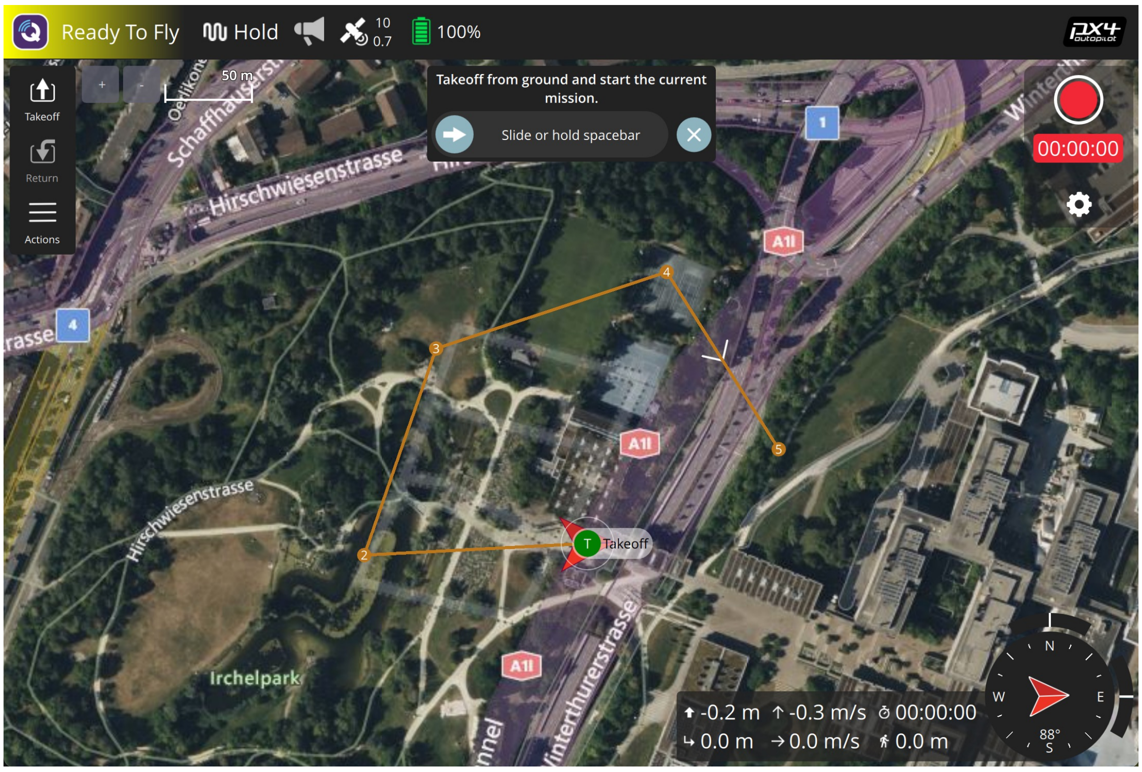


Figure 2.4: GUI of QGroundControl

operators to monitor the drone's status and environment in an intuitive way. Examples of widely used open-source GCS software include QGroundControl [44] and Mission Planner [39], both of which are compatible with autopilot systems. Figure 2.4 shows a screenshot of the QGroundControl interface, illustrating its key components, including a real-time map view, telemetry panel, flight mode selector, and mission status indicators. It displays the drone's location on a map, defines waypoints, adjusts flight parameters, and triggers specific actions such as payload deployment or camera operation. This interface greatly enhances situational awareness and simplifies mission execution, especially in complex or autonomous operations. In autonomous missions, the GCS plays a critical role in pre-mission configuration, in-flight supervision, and post-mission data analysis, ensuring safety, efficiency, and mission success. Furthermore, the GCS may serve as the gateway for multi-UAV coordination, data logging, and regulatory compliance in professional or commercial operations.

2.2.3 MAVLink

MAVLink [32, 36] is a lightweight, header-only communication protocol widely used in drone systems. It enables efficient and reliable message exchange between onboard flight controllers, GCS, and other companion computers. MAVLink messages are typically

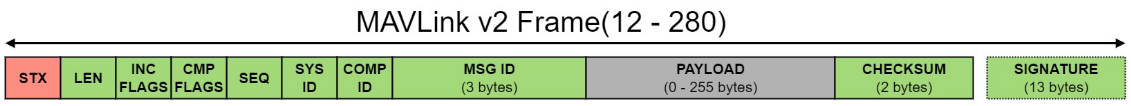


Figure 2.5: The MAVLink 2.0 protocol header

transmitted over serial or UDP/TCP links and are compatible with many autopilot systems, including PX4 and ArduPilot. MAVLink supports both telemetry data and control commands, making it an integral part of modern drone communication architectures. The protocol's extensibility and cross-platform support make it a key enabler for integrating third-party applications and performing closed-loop control.

Fig. 2.5 shows the header structure of MAVLink 2.0. MAVLink messages are structured into compact binary packets, which begin with a fixed-format header followed by a payload and checksum. The standard MAVLink 2.0 header consists of the following key fields:

- STX (1 byte): Start-of-frame marker of a MAVLink 2.0 packet with a fixed value (0xFD).
- LEN (1 byte): Length of the payload (0–255 bytes).
- INC FLAGS (1 byte): Bitmask indicating incompatible features with older MAVLink versions.
- CMP FLAGS (1 byte): Bitmask for backward-compatible feature flags.
- SEQ (1 byte): A monotonically increasing number for tracking lost packets.
- SYS ID (1 byte): ID of system (vehicle) sending the message. Used to differentiate systems on network (e.g., drone and GCS).
- COMP ID (1 byte): ID of component sending the message. Used to differentiate components in a system (e.g. autopilot and a camera).
- MSG ID (3 bytes): ID of message type in payload. Used to decode data back into message object.
- PAYLOAD (0-255 bytes): Message data. Depends on message type (i.e. Message ID) and contents.
- Checksum (2 bytes): CRC-16/MCRF4XX checksum calculated over the header and payload, used to detect transmission errors.

- Signature (13 bytes, optional): Optional digital signature for message authentication and tamper detection, used in secure communication scenarios.

Chapter 3

Related Work

In this chapter, we examine recent developments in joint drone-network simulation frameworks, which aim to provide realistic environments for evaluating autonomous drone operations under varying physical and network conditions. We also review existing research related to the Best-View Selection (BVS) and Next-Best-View Selection (NBVS) problems, with a focus on their application to modern 3D representations.

3.1 Drone Newtork Simulation Tools

3.1.1 Drone Simulator

AirSim [51], developed by Microsoft, is a widely adopted platform that provides photo-realistic visual environments and a range of virtual sensors for computer vision applications. It also includes a Python API for simulation control and data collection. However, AirSim employs a proprietary simplified drone dynamics model known as Fast Physics, which may not generalize well to scenarios requiring high-fidelity flight dynamics. Moreover, because AirSim's physics engine is tightly coupled with its rendering pipeline, the simulation capabilities are somewhat restricted in terms of modularity and scalability. In contrast, RotorS [16] and PX4 [37] are simulation frameworks built on Gazebo [31], a widely used robotic simulator in the research community. These platforms offer more realistic and modular drone dynamics models, supporting complex interactions through various sensor plugins, such as IMUs, GPS, and generic odometry sensors. With Gazebo's plugin-based architecture, these simulators are extensible to other types of robots, including ground and underwater vehicles. However, one notable limitation of Gazebo is its lack of support for photo-realistic rendering, which may limit its applicability in vision-based reconstruction or perception-driven tasks.

3.1.2 Network Simulator

NS-3 [47] is a discrete-event network simulator widely adopted for research and academic purposes, particularly in wireless communications and internet protocols. Designed with a modular architecture and implemented in C++ with Python bindings, NS-3 provides a rich set of models for simulating various network stacks, including LTE, Wi-Fi, 5G, and TCP/IP protocols. Its packet-level simulation capabilities allow researchers to analyze end-to-end latency, packet loss, routing behavior, and congestion under controlled conditions. OMNeT++ [57] is a general-purpose, component-based discrete-event simulation framework that is particularly well-suited for modeling communication networks, distributed systems, and multi-agent scenarios. It features a graphical runtime environment, an extensive model library, and a hierarchical module design that enables flexible system composition. OMNeT++ is highly extensible, making it a popular choice for simulating vehicular networks, IoT ecosystems, and increasingly, UAV communication systems. Compared to NS-3, OMNeT++ emphasizes simulation visualization and user interaction, which can facilitate debugging and educational use. While its networking models are not as extensive or low-level as NS-3's, OMNeT++ provides higher-level abstractions and intuitive configuration interfaces. Similar to NS-3, OMNeT++ lacks native support for drone dynamics, and thus requires integration with physical simulation tools for comprehensive drone system evaluation.

3.1.3 Joint Drone-Network Simulator

FlyNetSim [6] provides various network types and communication channels for drone communication simulations by integrating ArduPilot with NS-3. However, its user interface supports only a limited set of commands and does not include image capturing capabilities. UAVSim [23] is platform based on OMNeT++ that focuses on cyberattack simulation and security testing within drone networks. It allow research change the number of hosts and attackers, mobility models, and radio-propagation models. AirSim^N [55] is a co-simulation framework that integrates AirSim as the drone simulator and NS-3 as the network simulator. Built on top of NS-3, AirSim^N supports a wide range of network types, topologies, communication channels, and protocols, thereby enabling realistic simulation of drone communication systems. A middleware messaging system, ZeroMQ (ZMQ) [20], is employed to facilitate efficient and low-latency communication between the two simulators. FANS [12] is a co-simulation platform that integrates NS-3, Gazebo, and ROS to simulate both communication and mobility in ad-hoc networks. It enables multi-hop routing through intermediate nodes in the network simulator, making it suitable for applications such as post-disaster surveys. In conclusion, the simulators mentioned above do not encode messages using the MAVLink protocol during packet exchanges.

3.2 Scene Reconstruction

3.2.1 Non-neural 3D representations

BVS or NBVS problems have been studied in drone-assisted constructions of various 3D representations. For non-neural representations, Simultaneous Localization and Mapping (SLAM) systems [14, 15, 41, 42] simultaneously estimate the position of the robot and construct a map of the surrounding environment using visual or depth data. To enhance online 3D model reconstruction, a variety of trajectory planning algorithms have been proposed [7, 11, 53, 62] which guide drones toward unexplored regions in a 3D volumetric map to address the NBVS problem. For example, RH-NBV [7] proposes an algorithm to incrementally construct a random tree within the known free space and evaluate each branch based on its expected information gain. Song et al. [53] generate a coarse model before re-scanning for trajectory refinements. S-NBV [62] further improves the estimation of information gain by incorporating both volumetric and semantic maps to utilize richer contextual information. MAP-NBV [11] extends along this direction by introducing decentralized agents for collaborative 3D reconstruction. These methods [7, 11, 14, 15, 41, 42, 53, 62] primarily focus on geometric fidelity and scene coverage. They fall short in delivering photorealistic rendering quality, which is crucial for applications such as immersive visualization, virtual inspection, and digital twins.

3.2.2 NeRF representations

The BVS problem for constructing NeRF objects has also been studied. For example, ActiveNeRF [43] and Conditional-Flow NeRF [52] computed the estimated uncertainty levels of individual candidate views by applying the Bayesian rule on the current NeRF object so as to select the input views. Several follow-up studies considered the NBVS problem, For example, VBA [10] performs online 3D model reconstruction by incrementally integrating new images into a volumetric map that encodes color, density, and feature data per voxel. SRT [49] replaces the volumetric representation with a set-latent scene encoding by utilizing a transformer-based encoder-decoder architecture. NeU-NBV [25] incorporated a simulator to support an autonomous drone, where the simulator quantifies drone uncertainty levels of various candidate views, which may not be feasible in a real setup. In contrast, Zeng et al. published a series of papers solving the NBVS problem for NeRF objects [46, 63, 64]. In their first work, they estimated the uncertainty levels of candidate views and employed RRT (Rapidly-exploring Random Tree) to construct the drone trajectory [46]. Next, they switched to a more advanced A*-based algorithm to construct the drone trajectory so as to improve the overall data collection efficiency [63]. Last, they proposed methods to avoid local optima for better optimality and to support multiple drones for high efficiency [64]. These NeRF approaches [10, 25, 46, 49, 63, 64] rely on memory-intensive voxel-based representations, which may limit their scalability.

3.2.3 3DGS representations

To the best of our knowledge, only the BVS problem for 3DGS objects has been considered recently. For example, Savant et al. [50] introduced a framework integrating uncertainty estimation with 3DGS training using variational inferencing and a customized loss function. Their method significantly improved both uncertainty estimation accuracy and synthesized view quality, demonstrating good overall performance. Moreover, Jiang et al. [24] computed Fisher information of the radiance fields to solve the BVS problem using pixel-wise uncertainty quantification. Their method performs well in uncertainty quantification and view selection, showing its effectiveness in 3DGS construction.

The NBVS problem has recently garnered attention in the context of 3DGS reconstruction. Jin et al. proposed GS-Planner [26], a method that actively reconstructs 3DGS using a quadrotor. Their approach maintains and evaluates unobserved regions based on the current reconstruction quality to guide the robot in addressing the NBVS problem. This work was further extended by the same group [61] to improve exploration efficiency and mitigate the risk of local optima. Similarly, ActiveSplat [33] enhances exploration efficiency by incorporating a Voronoi graph into its planning framework. GS-SLAM [**?**] performs online 3DGS reconstruction and camera tracking by directly optimizing Gaussians and camera settings via differentiable rasterization. In contrast to these single-agent approaches, Zeng et al. [65] introduced a multi-robot autonomous framework for 3DGS reconstruction, aiming to reduce task completion time by leveraging cooperative agents. While these methods effectively explore unobserved regions in large-scale environments, they typically rely on memory-intensive 3D voxel maps, which may limit their scalability. Furthermore, their evaluations are primarily confined to indoor environments, and their generalize ability to outdoor or real-world scenarios remains largely unaddressed.

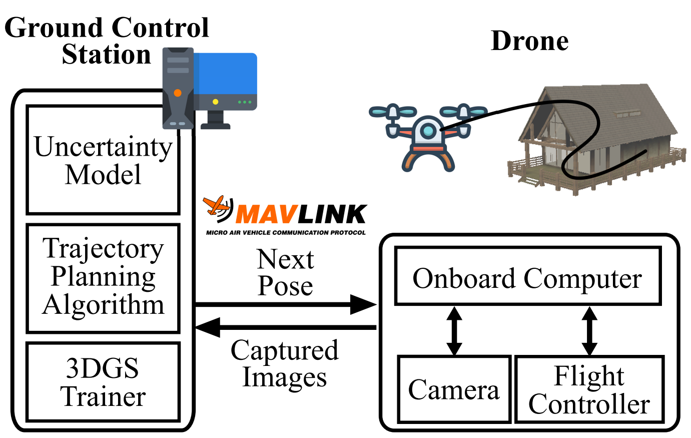
Differing from these works, the current paper is the first to solve the NBVS problem focus on visual quality for constructing 3DGS objects.

Chapter 4

Drone-Assisted 3DGS Construction

In this chapter, we present the considered scenario of drone-assisted 3DGS construction. We begin by outlining the overall system, including the role of the drone in data acquisition and trajectory planning. This is followed by a discussion of the 3DGS training procedure and the communication messaging protocol used to coordinate the drone's operations and data transmission.

4.1 System Overview



Application	MAVLink
Transport	UDP
Network	IP
Data Link	802.11 MAC
Physical	802.11g

Figure 4.1: Drone-assisted construction of a 3DGS object.

Fig. 4.1 reveals our considered scenario, which consists of three entities: *drone*, *Ground Control Station (GCS)*, and *edge server*. Here, the camera-equipped drone is responsible for following the instructions from the GCS, flying to the next position, and taking an image at the next orientation specified by the trajectory. To do so, the drone employs a flight controller to dynamically adjust the actuators, such as propellers, based on sensor readings, such as those from GPS readers and inertial sensors. The onboard computer collects all the sensors data, and is the main computation and communication unit of the drone. Whenever an image is captured, it is sent to the GCS, which computes the drone trajectory using a trajectory planning algorithm. This algorithm takes an incrementally built 3DGS object as input. The GCS also hosts an uncertainty model to estimate how much additional information a candidate pose brings to the current 3DGS object, which is crucial to trajectory planning algorithms.

More specifically, across recurring (disjoint) planning windows, the trajectory planning algorithm periodically produces a drone trajectory for the next planning window to cope with environment dynamics. Because constructing a 3DGS object is computation- and memory-demanding, GCS offloads the 3DGS trainer to an edge server. That is, the GCS passes all the captured views from the drone to the edge server as the input views for constructing a 3DGS object. The 3DGS trainer incrementally constructs a 3DGS object across multiple epochs, and sends the latest 3DGS object to the GCS upon request, which essentially is the time of executing the trajectory planning algorithm for the next planning window. Once the 3DGS construction process is over, the edge server outputs the latest 3DGS object.

4.2 3DGS Construction

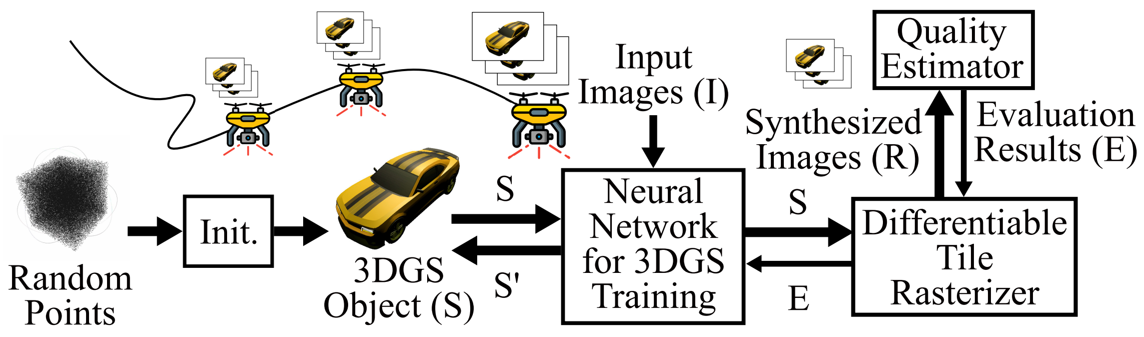


Figure 4.2: 3DGS trainer for incrementally generating a 3DGS object.

Fig. 4.2 zooms into the 3DGS trainer, which starts from a set of random points, serving as an initial 3DGS object $\boldsymbol{S}$. Next, the trainer uses the set of input images $\boldsymbol{I}$ and the current 3DGS object $\boldsymbol{S}$ to train a neural network to optimize the overall synthesized view quality. In particular, the 3DGS trainer rasterizes multiple synthesized images $\boldsymbol{I'}$ using $\boldsymbol{S}$ in order to evaluate $\boldsymbol{I'}$'s visual quality compared to the ground truth. The evaluation result E is used to update $\boldsymbol{S}$, turning it into the next 3DGS object $\boldsymbol{S}'$ before moving into the next epoch. Readers interested in more details on the 3DGS trainer are referred to Kerbl et al. [29].

4.3 Gaussian Construction Protocol

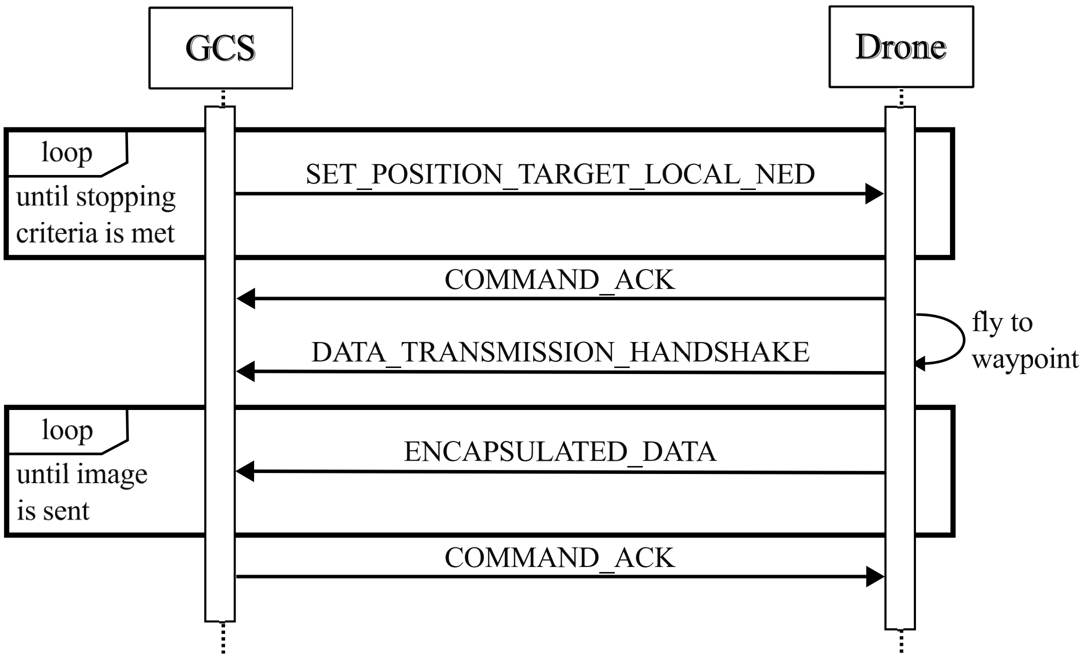


Figure 4.3: Sample operations of the proposed Gaussian construction protocol.

On top of MAVLink, we propose a Gaussian construction messaging protocol to enable the cooperation between the GCS and the drone. More specifically, Fig. 4.3 summarizes a sample exchange sequence of four key messages. The GCS first sends a *SET_POSITION_TARGET_LOCAL_NED* message to the drone, which contains the next waypoint. The drone responds with a *COMMAND_ACK* message and navigates to the waypoint. The drone then turns on the camera to capture an image. Next, the drone sends a *DATA_TRANSMISSION_HANDSHAKE* message to describe the image, which contains the image size, width, height, and the number of upcoming packets. This is followed by a series of *ENCAPSULATED_DATA* messages containing the image. After the GCS receives all messages, it replies with a final *COMMAND_ACK*. This completes a round of the online 3DGS refinement, and the GCS proceeds to the next refinement round.

Chapter 5

Trajectory Planning for 3DGS Construction

In this chapter, we formulate the trajectory planning problem, which is followed by optimal and efficient trajectory planning algorithms.

5.1 Design motivation

One key design decision of trajectory planning is how to model the uncertainty level, which can be done in multiple ways. We adopted an uncertainty model based on Fisher information [24] to evaluate each candidate pose's potential contribution to improve the current 3DGS object. We emphasize that Jiang et al. [24] solved the BVS problem, which is different from the NBVS problem we have in hand. Applying their solution to our scenario leads to two issues. First, their solution assumes the input views at all candidate poses have been captured, which is not the case in a real drone setup. Second, their solution does not allow for exploring new poses that are not in the set of input views, while ours offers much more freedom. With that said, planning a drone trajectory within a large search space is fundamentally challenging. To cope with it, we have to discretize the search space by sampling. We consider two alternative sampling methods: random and circular. The former sampling method is self-explanatory. The second method uniformly and sequentially samples the poses along the latitude, longitude, and radius centered at the 3D object. Each dimension comes with a minimum and a maximum, along with a sampling step; these parameters are user-specified.

5.2 Notations

Let $\boldsymbol{X}$ represent the set of candidate poses, with the first element of set x_1 representing the starting pose, and $M = |\boldsymbol{X}|$. At each pose, the position of the drone can be specified with latitude $\boldsymbol{\theta}$, longitude $\boldsymbol{\phi}$, and radius r in the polar coordinate system. The orientation of the drone camera points to the center of the 3D object. We use W to denote the duration of the planning window, which repeats N times until the construction is done. Hence, the total time duration is $W \cdot N$. Last, we let $\boldsymbol{H}$ represent the Fisher information of candidate poses in $\boldsymbol{X}$ given the current 3DGS object $\boldsymbol{S}$.

5.3 Formulation

For every pose $x_i \in \boldsymbol{X}$, the current 3DGS object $\boldsymbol{S}$ is used to generate a synthesized view $\tilde{y}_i$ to approximate the ground truth image y_i at x_i. Using $\tilde{y}_i$ and the current neural network model parameters, we compute the Fisher information $h_i \in \boldsymbol{H}$ [24]. Here, h_i is a positive real number, where a higher value indicates more additional information beyond $\boldsymbol{S}$.

With the above definition, our problem can be seen as a variant of the Traveling Salesperson Problem (TSP): given the planning window duration W and M candidate poses in $\boldsymbol{X}$, we select the optimal trajectory $\boldsymbol{P} = \{p_1, p_2, \ldots, p_N\}$ from $\boldsymbol{X}$ to maximize the total Fisher information within the planning window duration. Let $\boldsymbol{A}$ capture the expected moving time, i.e. $A[x_i][x_j]$ is the time for the drone to move from pose x_i to x_j. Let $\boldsymbol{V}$ be the decision variable, where boolean variable $v_{ij} = 1$ iff pose j is visited after i. Our problem can then be formulated as:

$$\text{maximize}_{\boldsymbol{P}} \textstyle\sum_{i=1}^{M} \sum_{j=1}^{M} h_i \cdot v_{ij} \tag{5.1a}$$

$$\text{subject to} : \textstyle\sum_{i=1}^{M} v_{ij} = \sum_{k=2}^{M} v_{jk} \; \forall j \in \{p_2, p_3, \ldots, p_{N-1}\} \tag{5.1b}$$

$$\textstyle\sum_{j=2}^{M} v_{1j} = 1; \tag{5.1c}$$

$$\textstyle\sum_{j=2}^{M} v_{ij} \leq 1; \forall i = 2, 3, \ldots, M; \tag{5.1d}$$

$$\textstyle\sum_{\forall v_{ij}=1} \boldsymbol{A}[\boldsymbol{x_i}][\boldsymbol{x_j}] \leq W; \tag{5.1e}$$

$$v_{ij} \in \{0, 1\}, \; \forall i, j = 1, 2, \ldots, M. \tag{5.1f}$$

The constraints in Eqs. (5.1b)–(5.1e) correspond to the flow conservation, starting pose, duplicated poses, and planning window duration, respectively.

5.4 Optimal algorithm

The trajectory planning problem in Eq. (5.1) is NP-Hard. We developed a Dynamic Programming (DP) algorithm [8] with a user-specified running time budget C, which returns

Algorithm 1 Dynamic Programming with Constraint: DPC$_C$

```
    Input: Time $\boldsymbol{A}$, planning window duration $W$, poses $\boldsymbol{X}$, object $S$, running time budget $C$
    Output: Trajectory $\boldsymbol{P^*}$
 1: $dic \leftarrow \emptyset$; $hp$.push$(0, 0, 0, [0])$; $u^* \leftarrow 0$; $\boldsymbol{P^*} \leftarrow [0]$; $\boldsymbol{U} \leftarrow U_o(\boldsymbol{X}, S)$
 2: while $hp$ is not empty do
 3:     if $t_p > C$ then break
 4:     $(u, t, x, \boldsymbol{P}) \leftarrow hp$.pop()
 5:     if $u \geq u^*$ then $u^* \leftarrow u$; $\boldsymbol{P^*} \leftarrow \boldsymbol{P}$
 6:     for $i \leftarrow 1$ to $M$ do
 7:         if $x_i \notin \boldsymbol{P}$ then
 8:             $t' \leftarrow t + A[x][x_i]$
 9:             if $t' \leq W$ then $u' \leftarrow u + \boldsymbol{U}[x_i]$; $\boldsymbol{P'} \leftarrow \boldsymbol{P} + \{x_i\}$
10:                 if $(x_i, t') \notin dic$ or $dic[(x_i, t')] < u'$ then $dic[(x_i, t')] \leftarrow u'$; $hp$.push$(u', t', x_i, \boldsymbol{P'})$
11: Return $\boldsymbol{P^*}$
```

the best-known solution after C. We refer to this algorithm as Dynamic Programming with Constraint, or DPC$_C$, where C is the running time budget; we note that DPC$_\infty$ degrades to the optimal DP algorithm. To quantify the contribution brought by each candidate pose $x_i \in \boldsymbol{X}$, given the current $\boldsymbol{S}$, we define a utility function $U_o(x_i, \boldsymbol{S})$ using its Fisher information, i.e:

$$U_o(x_i, S) = h_i. \tag{5.2}$$

DPC employs several intermediate data structures: (i) dictionary dic stores the maximum utility of each DP state, (ii) max-heap hp sorted by their utility values (denoted as u), along with total elapsed time t, poses x, and trajectory $\boldsymbol{P}$, (iii) best-known total utility u^*, (iv) best-known trajectory $\boldsymbol{P^*}$, and (v) running time t_p.

Algorithm 1 gives the pseudocode of DPC$_C$. The algorithm begins with pushing the initial tuple of total utility, elapsed time, pose, and trajectory onto hp. Line 2 ensures the while loop continues until hp is empty. At each iteration for this loop, we extract and process the next tuple from hp. Line 5 updates u^* and $\boldsymbol{P^*}$ if the current utility exceeds the best-known u^*. The for-loop starts from line 6 and evaluates all possible candidate poses x_i. Line 7 confirms that x_i has not yet been visited. The time to move to x_i is added to the current elapsed time in line 8. Line 9 adds the utility of x_i to the total utility and appends x_i to the trajectory if the elapsed time does not exceed W. Line 10 stores the current state in dic and pushes the current tuple onto hp if it is a new state or the next total utility surpasses the state's utility value in dic. The algorithm returns the best-known trajectory $\boldsymbol{P^*}$ in line 11. The time complexity of this algorithm is $O(M^2 2^M \log M)$, where DP accounts for $M^2 2^M$ and heap accounts for $\log M$.

Algorithm 2 A*-inspired Utility Maximization: AUM

```
    Input: Time $\boldsymbol{A}$, planning window duration $W$, poses $\boldsymbol{X}$, object $S$
    Output: Trajectory $\boldsymbol{P}^*$
1: $\boldsymbol{P}^* \leftarrow [0]$
2: $t \leftarrow 0$
3: while $t \leq W$ do
4:     for $i \leftarrow 1$ to $M$ do
5:         for $j \leftarrow 1$ to $M$ do
6:             if $i \neq j$ and $x_i, x_j \notin \boldsymbol{P}^*$ and $t + A[x_i][x_j] \leq W$ then store $U_A(x_i, x_j, S, t)$ in $\boldsymbol{U}$
7:     $x_{i^*} \leftarrow \arg\max_{x_i} \boldsymbol{U}$
8:     $\boldsymbol{P}^*$.append($x_{i^*}$); $t \leftarrow t + A[\boldsymbol{P}^*.\text{last()}][x_{i^*}]$
9: return $\boldsymbol{P}^*$
```

5.5 Efficient algorithm

We also propose an efficient algorithm inspired by the A* algorithm [13] when real-timeness is crucial. We call it the A*-inspired Utility Maximization (AUM) algorithm. As a greedy algorithm, AUM jointly considers the next two poses x_i and x_j beyond a given trajectory $\boldsymbol{P}^*$. To quantify the potential contribution of adding pose x_i to $\boldsymbol{P}^*$, with the total elapsed time denoted as t, we define a utility function $U_A(x_i, x_j, S, t)$ as:

$$U_A(x_i, x_j, S, t) = h_i + h_j / A[x_i][x_j] \cdot [(W - (t + A[x_i][x_j])]), \tag{5.3}$$

where the first term represents the Fisher information provided by pose x_i, and the second term represents the weighted Fisher information provided by pose x_j. In particular, the weighted Fisher information is: (i) inversely proportional to the moving time from pose x_i to x_j, and (ii) proportional to the remaining time in the planning window duration after reaching pose x_j.

Algorithm 2 gives the pseudocode of the proposed AUM algorithm. Lines 3 and 6 ensure that the resulting trajectory can be completed within the duration of the planning window. The for loops, starting from lines 4 and 5, iterate through all candidate poses x_i and x_j. After storing the utility function value of all combinations of x_i and x_j, line 7 finds the next two candidates' poses that lead to the highest utility function value defined in Eq. (5.3). Line 8 appends pose x_i to trajectory $\boldsymbol{P}^*$, and line 9 returns the best-known $\boldsymbol{P}^*$. The time complexity of the AUM algorithm is $O(M^3)$, as the algorithm selects at most M poses, and the time complexity for each pose is $O(M^2)$.

Chapter 6

Experiments

In this chapter, we evaluate the effectiveness of our proposed trajectory planning algorithms, DPC and AUM, using the developed drone simulator. The primary objectives of our experiments are threefold: (i) to quantify the time savings enabled by on-the-fly trajectory optimization and model training, (ii) to assess the improvement in visual quality of the reconstructed 3DGS objects, and (iii) to analyze the impact of key parameters on overall system performance.

6.1 Implementations

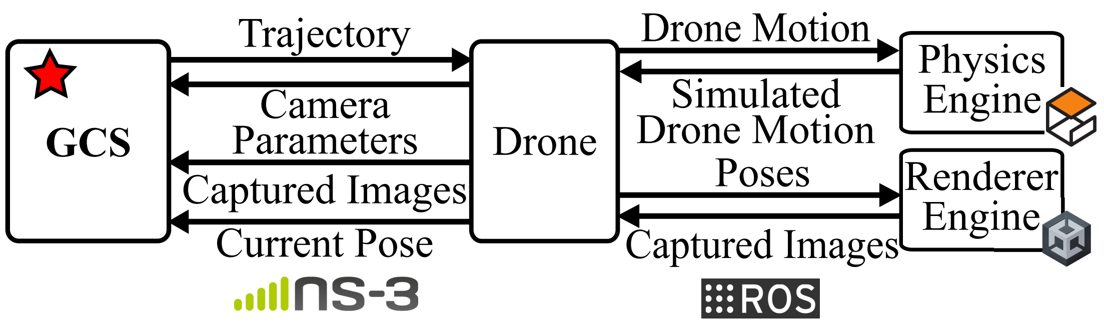


Figure 6.1: The architecture of our 3DGS capturing testbed.

We implemented a detailed testbed to evaluate our proposed solution. Fig. 6.1 gives the main components of our simulator:

- **ROS.** Robot Operating System (ROS) [45] is used to enable modular, real-time communication through a publish–subscribe mechanism.

- **Physics engine.** Gazebo [31] is used to capture the drone's physical behavior, including aero-dynamics, sensor readings (e.g., GPS, IMU), and environments (e.g., wind and collisions).

- **Renderer engine.** Unity [2] is used for realistically synthesized images, thanks to its dynamic lighting and shadow casting.

- **Network simulator.** NS-3 [47] is used to perform the packet-level simulations between the GCS and the drone. It supports multiple wireless interfaces (e.g., Wi-Fi, LTE, D2D), and models real-world factors such as network congestion, environmental interference, and signal attenuation through obstacles. Additionally, we extend NS-3 with MAVLink protocol [**?**] to simulate packet-level telemetry and control message exchange between a GCS and a drone.

Combining all these components, our simulator is able to support high-fidelity visualization and capture realistic network communications. Specifically, our drone simulator offers the following capabilities: (i) realistic physical effects, (ii) photorealistic rendering, and (iii) packet-level communication.

We built our testbed upon several open-source projects. For instance, the simulator structure is based on the open-source FANS simulator [12] using a combination of C/C++ and C#. Moreover, we use a widely adopted autopilot software stack, PX4 [37], to capture drone dynamics. Furthermore, we employ Gazebo [31] for physics-based simulations. Unity [2] performs photorealistic rendering, while FlightGoggles [18] enables seamless interaction between PX4 and Unity. For communication, we adopt the official MAVLink C library [**?**], allowing us to encode and decode messages in the simulator. Fig. 6.2 shows screenshots of the physics and renderer engines of our simulator.

6.2 Flow-Level Simulation Setup

To isolate and evaluate the effectiveness of our trajectory planning algorithm without interference from communication-related factors, we conducted a set of experiments under idealized network conditions. We assume a perfect communication channel between the GCS and the drone, where all command and data packets are transmitted instantaneously and without loss. Specifically, we disabled the ns-3 network module and bypassed any network-induced latency, jitter, or packet drop.

As the NBVS problem for 3DGS objects has never been studied, we implemented two baseline algorithms: (i) *Sequential (SEQ)*, which constructs the 3DGS object using all captured views, and (ii) *Adjusted Sequential (ASEQ)*, which employs Fisher information [24] to select representative input views with a number capped to the maximum between those of DPC and AUM (for a comparable workload of the 3DGS trainer). The edge server hosts a 3DGS trainer enhanced by Kerbl et al. [29]. In total, we modified or added 44 files with 7014 line-of-code changes in our testbed. We adopted three synthetic objects in the experiment, *Car* [1], *Ship* [54], and *Cabin* [5]. Each object was placed in a $4 \times 2 \times 2\ m^3$ bounding box with a direct light and an environmental light with a white background. We ran our experiments on a workstation with an AMD Ryzen 7 5700X CPU, 32 GB of RAM, and an NVIDIA RTX 3090 Ti GPU. In each simulation run, the drone flew for 300 s. After that, the edge server continued training the 3DGS object for 50 seconds. For sampling candidate poses, we set $r \in [4, 10]$ to random sampling; and $\boldsymbol{\theta} \in \{15, 30, 45, 60\}$, $\boldsymbol{\phi} \in \{0, 20, \ldots, 340\}$, and $r \in \{4, 5, \ldots, 10\}$ to circular sampling. We varied $W \in \{25, \underline{50}, 75, 100\}$, $M \in \{5, \underline{10}, 20, 40, 80\}$, and $C \in \{0.25, 0.5, \underline{1}, 4\}$, where the underlined values are the defaults. We report the average performance results from five simulations with 95% confidence intervals whenever possible.

- Hyperparameters:
  - $W \cdot N = 300$
  - Additional training time $= 50$
  - Random method:
    - $r \in [4, 10]$
  - Circular method:
    - $\boldsymbol{\theta} \in \{15, 30, 45, 60\}$
    - $\boldsymbol{\phi} \in \{0, 20, \cdots, 340\}$
    - $r \in [4, 10]$
- Experiment variable:
  - $W = \{25, \underline{50}, 75, 100\}$
  - $M = \{5, \underline{10}, 20, 40\}$
  - $C = \{0.25, 0.5, \underline{1}, 4\}$

6.3 Flow-Level Simulation Results

Merits of our trajectory planning algorithms. We start with a sample run of our experiments. Fig. 6.3 shows sample results of Cabin with random sampling from different trajectory planning algorithms. Fig. 6.3(a) depicts the visual quality over time. In the first 300 seconds, our algorithms can steadily improve the quality of the 3DGS object as we solve the NBVS problem, allowing us to refine the 3DGS object on the fly instead of after all views are captured. We observe some fluctuations in the quality of DPC and AUM, which can be attributed to the 3DGS mechanism that periodically removes some 3D Gaussian to prevent excessive 3D Gaussian density. In Fig. 6.3(b), we show a synthesized view of the final 3DGS object from SEQ. We also give the corresponding synthesized view of the 3DGS object at the 140-th second from our DPC in Fig. 6.3(c), which clearly has a much higher perceived quality compared to Fig. 6.3(b). A deeper investigation revealed that DPC provided fairly good visual quality in a short time and continued optimizing the 3DGS object. Fig. 6.3(a) also depicts that DPC outperformed SEQ by 1.85 dB and ASEQ by 8.01 dB in PSNR when the final 3DGS objects were constructed. Fig. 6.4 presents sample results of Car with circular sampling from different trajectory planning algorithms. Observations on the figure are in line with those made on Fig. 6.3: DPC delivered comparable synthesized visual quality to that of SEQ at the 174-th second, and boosted the final visual quality by up to 5.90 dB. This performance boost of DPC was realized with fewer input views: from 52 (SEQ) to 31 (DPC), showing the strength of an optimized drone trajectory.

Next, we present the results from five runs and across different 3D objects. Fig. 6.5 gives the results from random sampling. Compared with SEQ, our proposed DPC achieved a better visual quality at 22.17 dB in PSNR, which can be attributed to 13.6 more input views enabled by DPC's optimized flying trajectory. Fig. 6.6 gives the results from circular sampling. The results show that SEQ captured 21.0 more input views than DPC on average. Nonetheless, DPC still outperformed SEQ by 1.60 dB in PSNR on average. This is because our algorithms carefully select the most promising candidate poses based on their utility function values. Adding to that, because we solved the NBVS problem, our algorithms train 3DGS objects on the fly, allowing them to get more complete 3DGS objects sooner.

Impacts of configurable parameters. We first varied the planning window duration W between 25 and 100 s. Fig. 6.7 reports the performance under different W. Fig. 6.7(a) depicts that the visual quality is not affected by W with random sampling. Fig. 6.7(c) shows that the visual quality generally decreases with larger W values under circular sampling. When increasing W from 25 to 100, with the DPC and AUM algorithms, the overall quality drops by 2.47 and 2.96 dB in PSNR, respectively. Figs. 6.7(b) and 6.7(d) reveal that the numbers of input views decrease when W is increased. There are a few possible causes. First, theoretically, when W is sufficiently large, our algorithms get a chance to select the most suitable poses from the candidate poses for a good trajectory. However, in practice, for the DPC algorithm, a longer planning duration leads to higher computational complexity. For example, in our experiments, when W exceeds 75 s, DPC cannot complete the optimal trajectory with smaller C, forcing it to return the best-known solution. In contrast, AUM does not encounter this issue; even with $W = 100$ s, AUM takes only 0.332 milliseconds on average to plan the trajectory. Second, a large W reduces the flexibility of both DPC and AUM in trajectory planning. Given a limited total time, a larger W value results in fewer re-planning opportunities, thereby decreasing the chances for our algorithm to adjust according to the utility of existing 3DGS objects.

Next, we vary the running time budget C. Fig. 6.8(a) shows the visual quality as the value of C increases. Since AUM needs a much shorter running time than any of the considered running time budgets, varying C has a negligible impact on its performance. Hence, we focus on discussing C's impact on DPC. We take Cabin as an example. When increasing C from 0.25 to 4 s, the number of input views is reduced from 30 to 26, with a 1.14 dB boost to PSNR. Fig. 6.9 depicts the overall results. On average, increasing C from 0.25 to 4 s yields a 0.226 dB improvement in quality and reduces the required number of input views by 1.67. The reason is that when our trajectory planning algorithms have time, they can find the best trajectory in time. The drone then follows the trajectory for higher utility and fewer input views. However, the implication of C is also highly related to the decision of W, which determines whether the drone has enough planning and capturing time. Thus, we recommend adaptively choosing the C for DPC and W for DPC and AUM based on usage scenarios and computing resources.

Fig. 6.10 shows the results of increasing the number of candidate poses from 5 to 80. We observe that, as M increases, the visual quality improves. The reason is that larger M increases the number of candidate poses for the trajectory planning algorithms for selection so that our algorithms can select better trajectories. However, when we increase M beyond 20, DPC may use up all the running time budget C. Hence, the visual quality does not significantly improve once M goes beyond 20. Fig. 6.11 presents the quality difference between the random and circular sampling. This figure shows that with larger W, random sampling leads to better visual quality because of the randomness. In contrast, when W is small, the candidate poses that are far away may miss the opportunity to be selected. Hence, randomness no longer helps.

6.4 Packet-Level Simulation Setup

In contrast to the idealized setting, we also conducted experiments under realistic network conditions to assess the robustness and reliability of our protocol and trajectory execution in the presence of communication disturbances. In this configuration, we enabled the ns-3 network simulation module to introduce controlled network impairments. These impairments were designed to emulate typical wireless communication scenarios encountered in real-world drone deployments.

The trajectory planning setup in the packet-level simulation is identical to that of the flow-level simulation, ensuring a consistent comparison baseline across both experimental conditions. However, unlike the idealized network assumed in the flow-level case, this setting incorporates realistic network behaviors. Specifically, we integrate the Building and Propagation Loss modules [3] into NS-3. The material properties were defined as "Metal" for Car and Ship, and "Wood" for Cabin. The drone has the same initial position in all experiments.

Within the GCS, we implement two trajectory planning algorithms: (i) *DPC* and (ii) *AUM* to identify waypoints for the sake of demonstrations. We also implement offline *(OF)* reconstruction as the baseline, which reconstructs with all captured images. Each online reconstruction task is allocated 300 seconds, during which the system simultaneously captures images and progressively reconstructs the 3DGS model. The offline reconstruction baseline first completes the image capture phase entirely before commencing the 3D reconstruction. We adopt the IEEE 802.11g Wi-Fi standard using the IP protocol stack. Message transmission utilizes UDP in conjunction with the MAVLink 2.0 protocol.

For online reconstruction (*DPC* and *AUM*), camera poses are randomly sampled within a spherical coordinate space. Specifically, the radial distance r is sampled from the range from $\{4, 7, 10\}$, the elevation angle θ from $[15°, 90°]$, and the azimuthal angle ϕ from $[0°, 360°)$. For offline reconstruction (*OF*), four fixed circular trajectories are predefined around the 3D object. The camera poses along this trajectory are constrained to a constant elevation angle $\theta = 15°$, while the azimuthal angle ϕ is uniformly sampled over the range $[0°, 360°)$. Four circular orbits are discretized into 3, 6, 12, or 24 evenly spaced candidate poses. The JPEG compression ratio is varied across quality levels: $Q \in \{100, \underline{80}, 60, 40, 20\}$. We let $Q = 80$ if not otherwise specified. The distance between the GCS and the building is $D = 50$. All results are evaluated on a test set of 100 images and are reported as averages, accompanied by 95% confidence intervals where applicable.

- Hyperparameters:
  - $W \cdot N = 300$
  - Additional training time $= 0$
  - $D = 50$
  - Online reconstruction:
    - $r \in \{4, 7, 10\}$
    - $\boldsymbol{\theta} \in [15°, 90°]$

    - $\phi \in [0°, 360°)$
  - Offline reconstruction:
    - $r \in \{4, 7, 10\}$
    - $\boldsymbol{\theta} = 15°$
    - $\phi \in [0°, 360°)$
- Experiment variable:
  - $Q = \{100, \underline{80}, 60, 40, 20\}$

6.5 Packet-Level Simulation Results

Merits of online reconstruction. We first present a sample run of our experiments in Fig. 6.12, which shows that the visual quality of the 3DGS model produced by online reconstruction improves over time. In contrast, the offline reconstruction generates a single 3DGS model after its training is completed. We make three observations on the figure. First, the DPC and AUM algorithms generally outperform OF. Second, enough number of candidate poses (say 24) is needed for good reconstructed quality. Lastly, a radial distance of 4 gives better visual quality than 7 and 10. *Such observations are not possible without our physics simulator.*

Impacts of different objects. Fig. 6.13 reports sample performance with 24 candidate poses from different trajectory planning algorithms and diverse objects. We observe that the performance trends across the algorithms are consistent among objects. In particular, Figs. 6.13(a) and 6.13(b) show that DPC and AUM achieve better visual quality, thanks to more input images. For example, on average (across three objects), DPC boosts the visual quality of OF:4 by 2.49 dB, at an expense of 8.75 more images. Fig. 6.13(c) reports the uplink throughput, which shows that OF leads to much more bursty traffic. A closer look indicate that unlike the DPC and AUM algorithms that would intelligently fly to the next best pose, the OF trajectories sequentially capture a series of images in close spatial (and temporal) proximity. This shows DPC and AUM also incur lower network traffic burst-ness. *Such observations are not possible without our photorealistic simulator.*

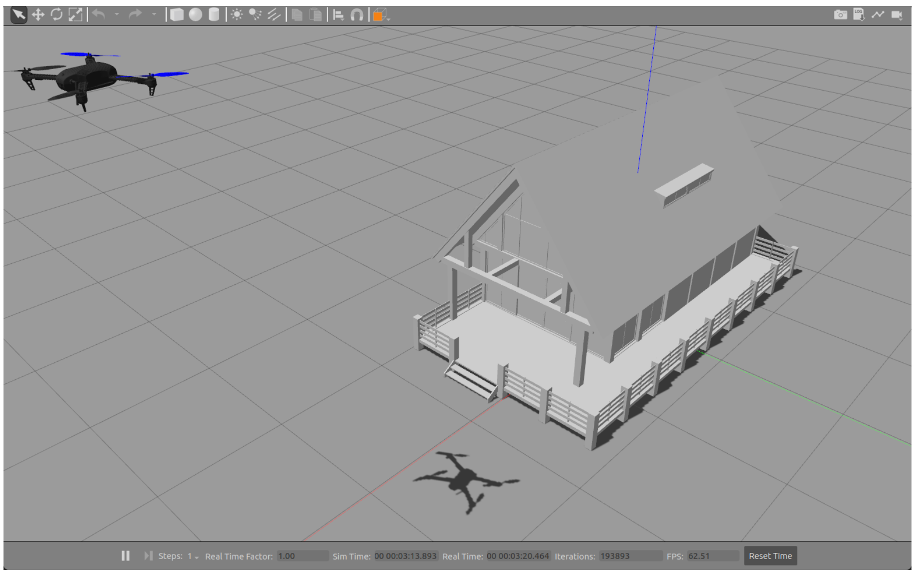
Impacts of JPEG quality levels. Fig. 6.14 reports the tradeoff between visual quality and unlink throughput using the JPEG quality level Q as the control knob. Sample results from 24 candidate poses are given here. For brevity, we omit curves from OF:8 and OF:10. Also we turn their x-axis in log-scale due to wider throughput range. We observe that higher Q levels do not necessarily result in higher visual quality. This can be attributed to the fact that the bandwidth between a GCS and a drone is limited. Because higher Q

Table 6.1: Resource Utilization in Online Reconstruction

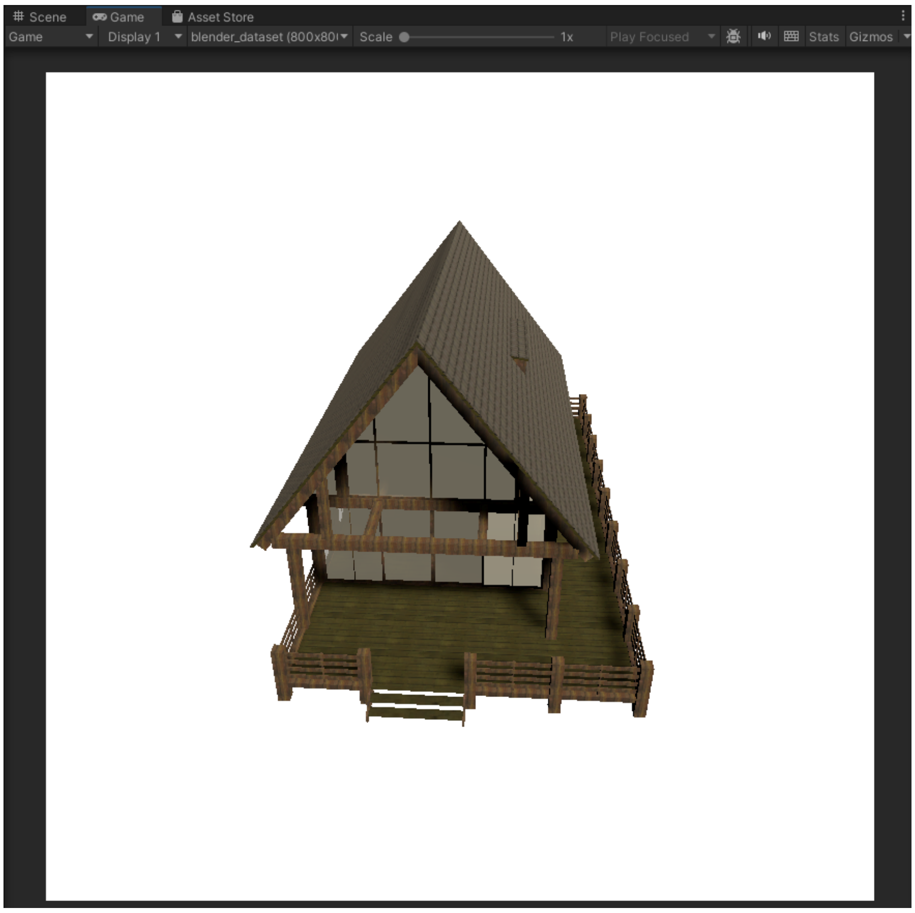
	Ship	Car	Cabin
	DPC		
CPU utilization	26.97 ± 3.04	24.41 ± 2.55	36.63 ± 2.41
GPU utilization	84.59 ± 6.82	85.32 ± 6.38	87.40 ± 7.55
Memory	63.07 ± 0.41	58.30 ± 0.37	62.06 ± 0.18
GPU memory	10.10 ± 0.44	9.56 ± 0.52	9.75 ± 0.50
	AUM		
CPU utilization	26.24 ± 3.90	26.22 ± 2.63	37.85 ± 2.89
GPU utilization	84.25 ± 8.02	85.91 ± 6.30	87.14 ± 10.28
Memory	62.77 ± 0.29	63.82 ± 0.14	62.02 ± 0.15
GPU memory	10.11 ± 0.41	14.52 ± 0.64	10.64 ± 2.84

levels result in bigger images (thus, higher throughput), fewer images can be transmitted to the GCS before the simulation is over. *Such observations are not possible without our packet-level simulator.*

Resource Utilization. Tab. 6.1 reports the average resource utilization of DPC and AUM algorithms across three different scenes. GPU resources are primarily consumed during the training process of the 3DGS reconstruction, while CPU resources are used for tasks such as message decoding and planning. Overall, GPU utilization remains consistently high (above 84%) across both algorithms and all scenes, indicating the computational intensity of online 3DGS reconstruction. CPU utilization varies slightly, with the Cabin scene showing higher usage due to its structural complexity. Memory and GPU memory consumption are relatively stable. Despite these differences, both approaches maintain moderate resource usage, confirming the feasibility of deploying online reconstruction under realistic computational constraints.

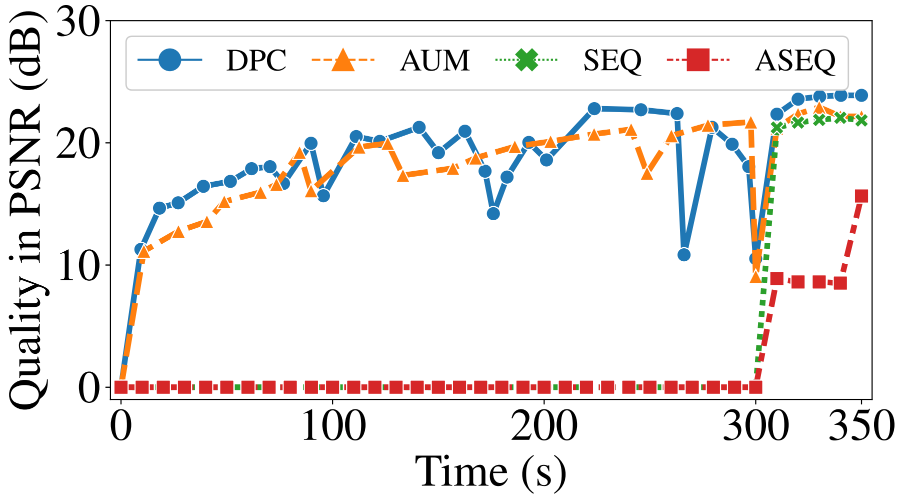


(a)

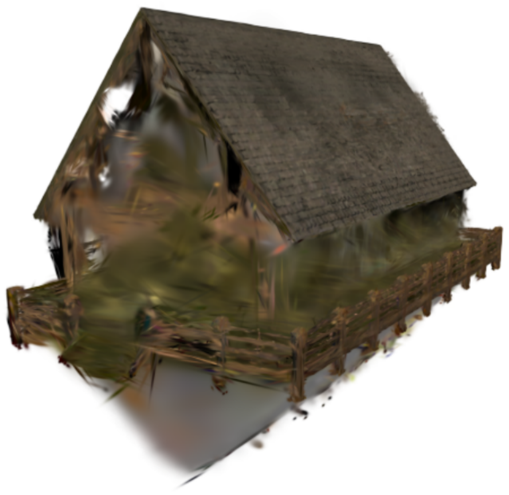


(b)

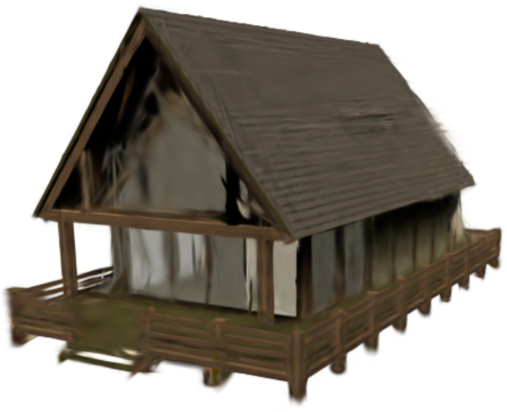
Figure 6.2: Screenshots of our FlyGS simulator in: (a) Gazebo and (b) Unity GUIs.



(a)

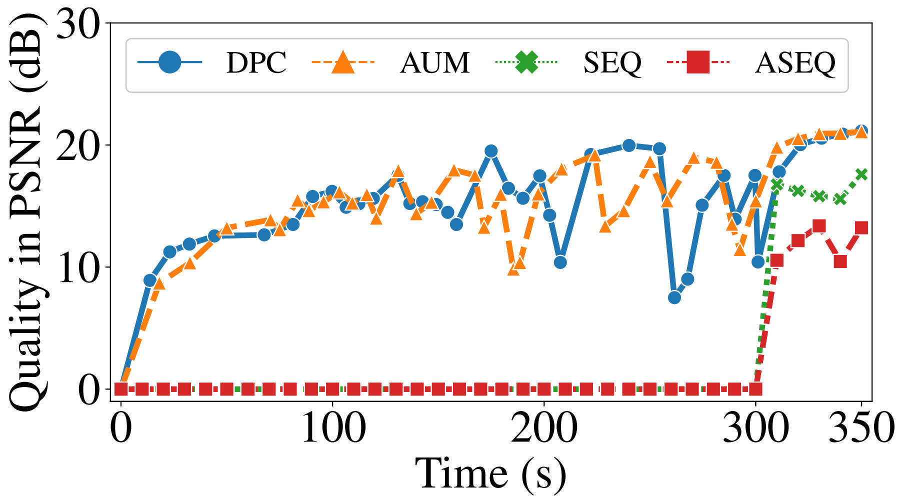


(b)

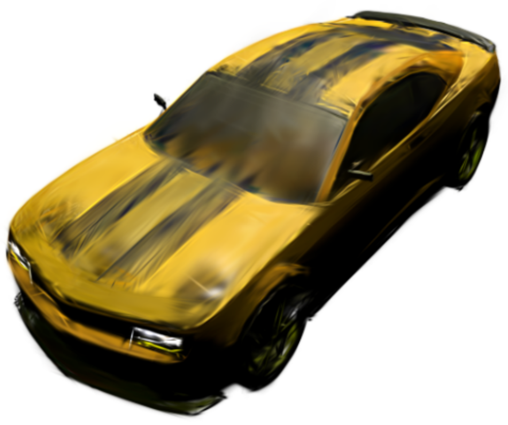


(c)

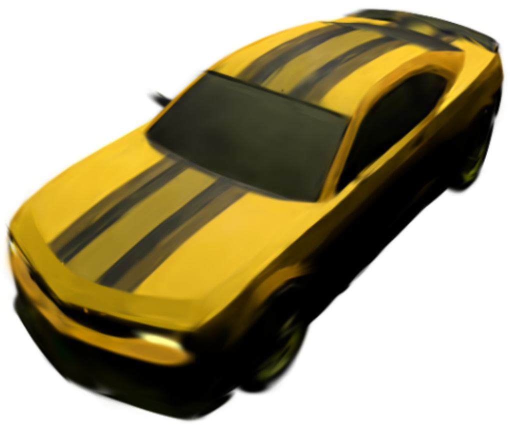
Figure 6.3: Performance comparison of different algorithms under default settings: (a) quality in PSNR, (b) sample synthesized view from SEQ, and (c) from DPC at the 140-th second. Sample results from Cabin and random sampling are shown.



(a)

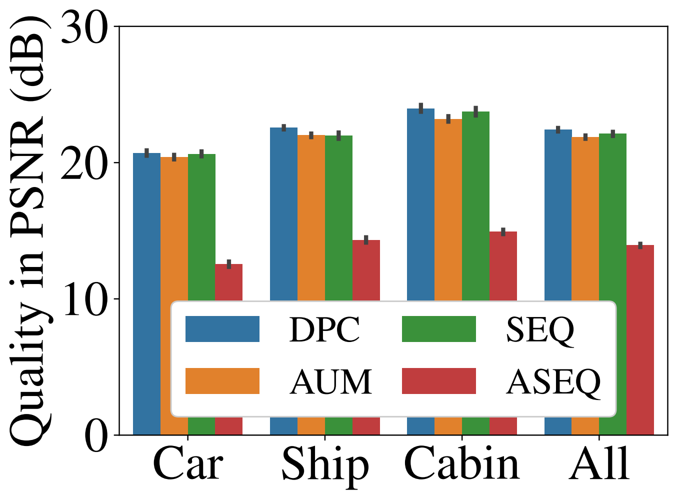


(b)

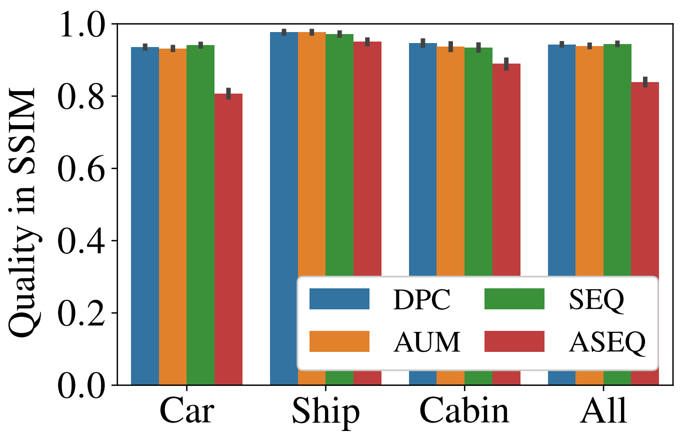


(c)

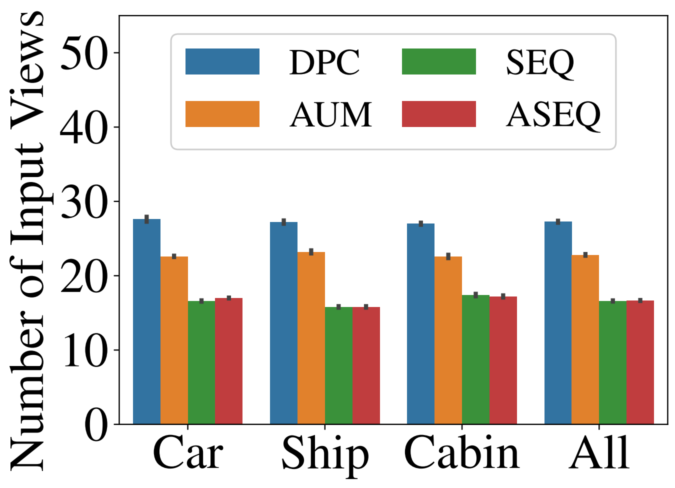
Figure 6.4: Performance comparison of different algorithms under default settings: (a) quality in PSNR, (b) sample synthesized view from SEQ, and (c) from DPC at the 174-th second. Sample results from Car and circular sampling are shown.



(a)

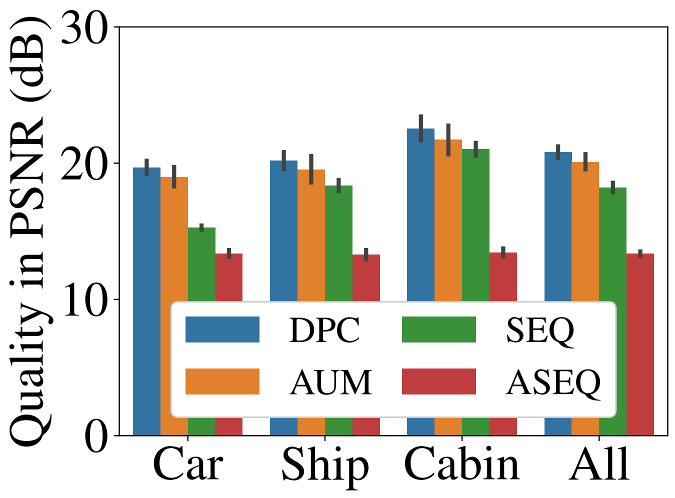


(b)

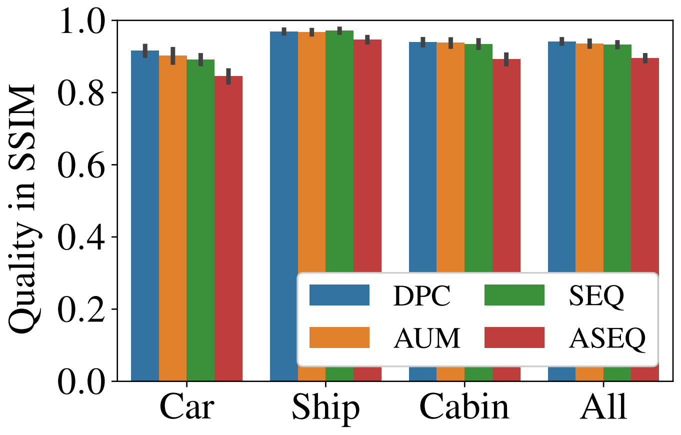


(c)

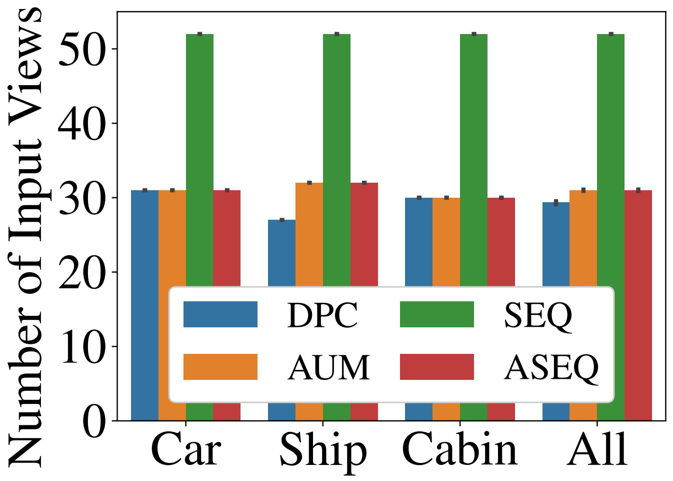
Figure 6.5: Performance of different algorithms with random sampling: (a) quality in PSNR, (b) quality in SSIM, and (c) numbers of input views.



(a)

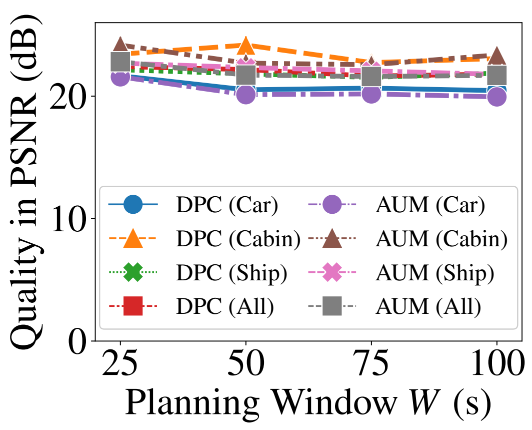


(b)

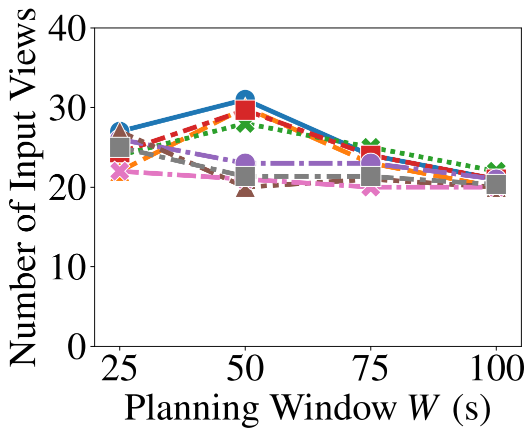


(c)

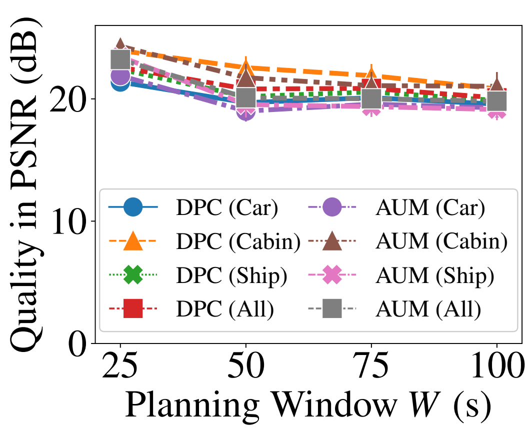
Figure 6.6: Performance of different algorithms with circular sampling: (a) quality in PSNR, (b) quality in SSIM, and (c) numbers of input views.



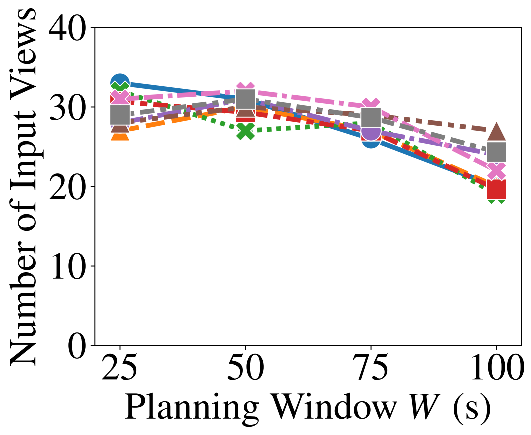
(a)



(b)

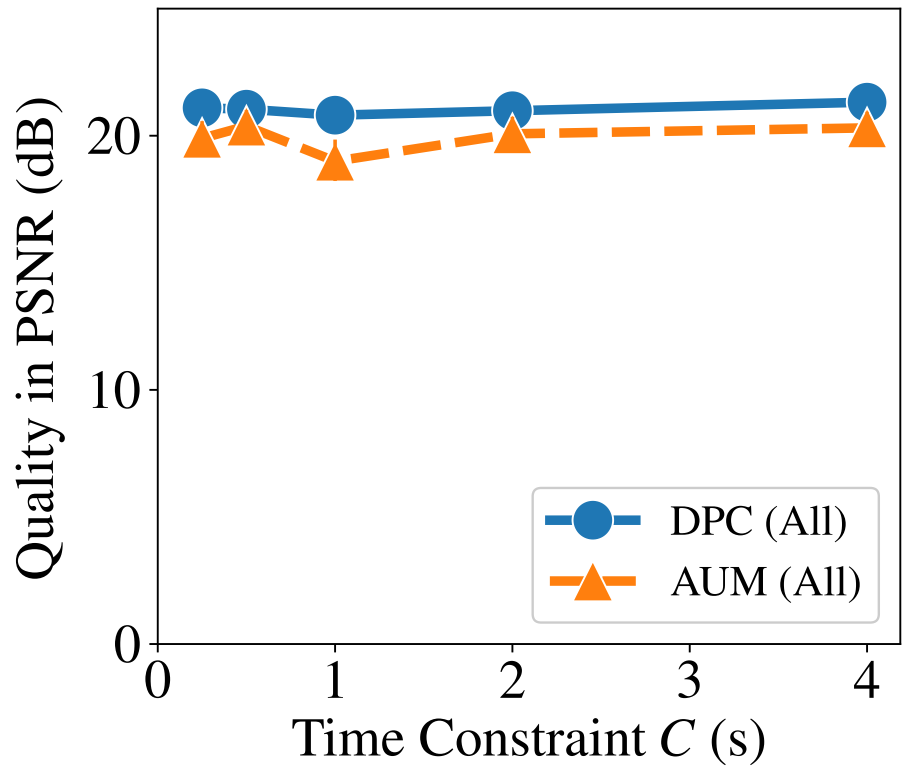


(c)

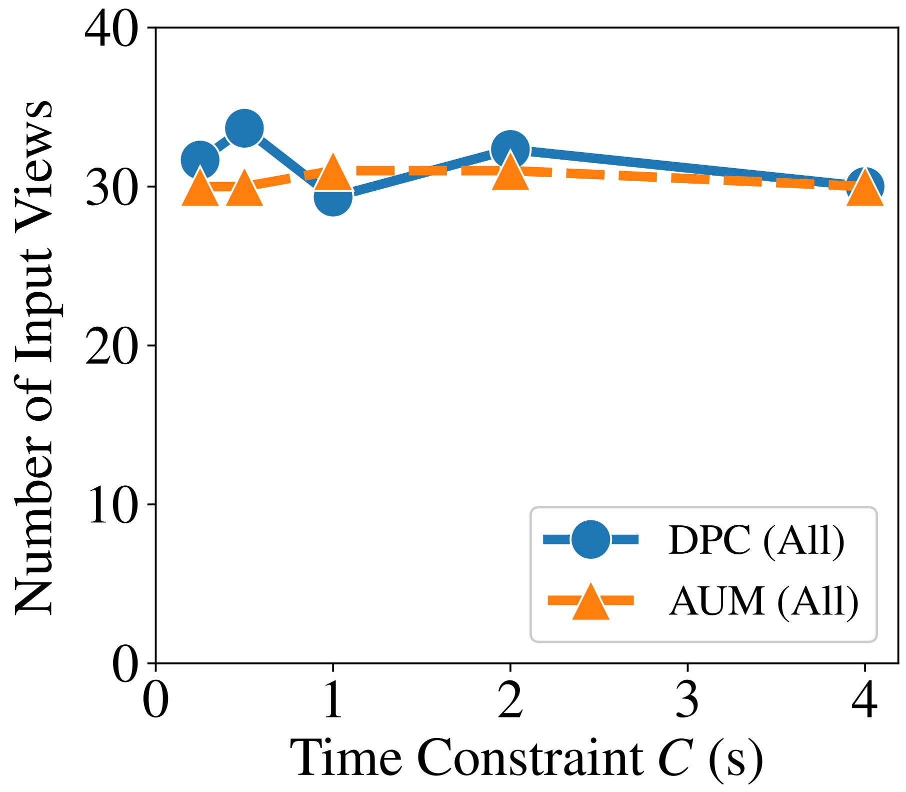


(d)

Figure 6.7: Performance under different W: (a), (c) quality in PSNR and (b), (d) number of input views. (a), (b) are from random and (c), (d) are from circular sampling.

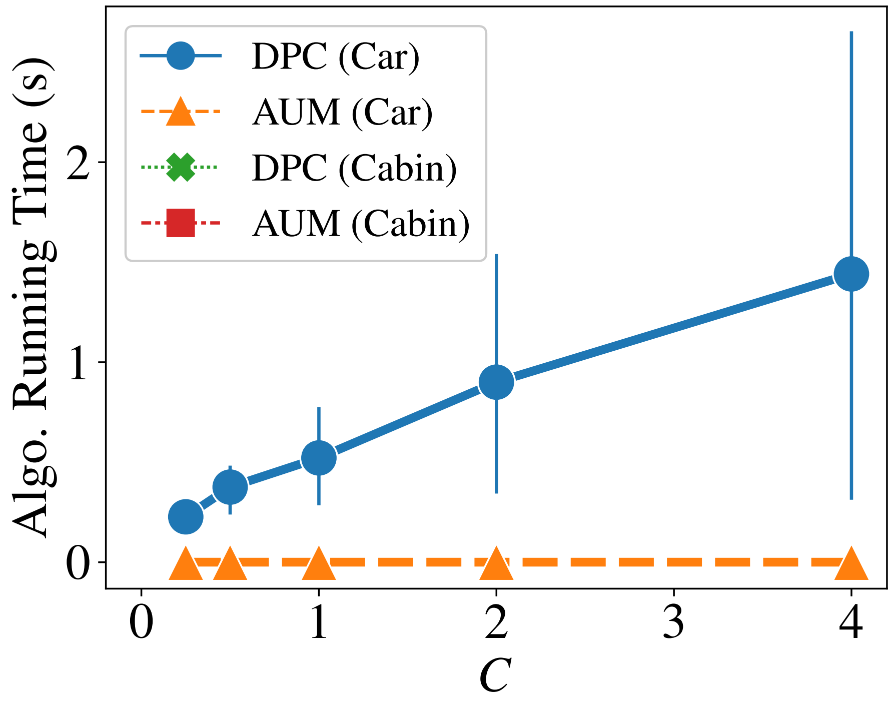


(a)

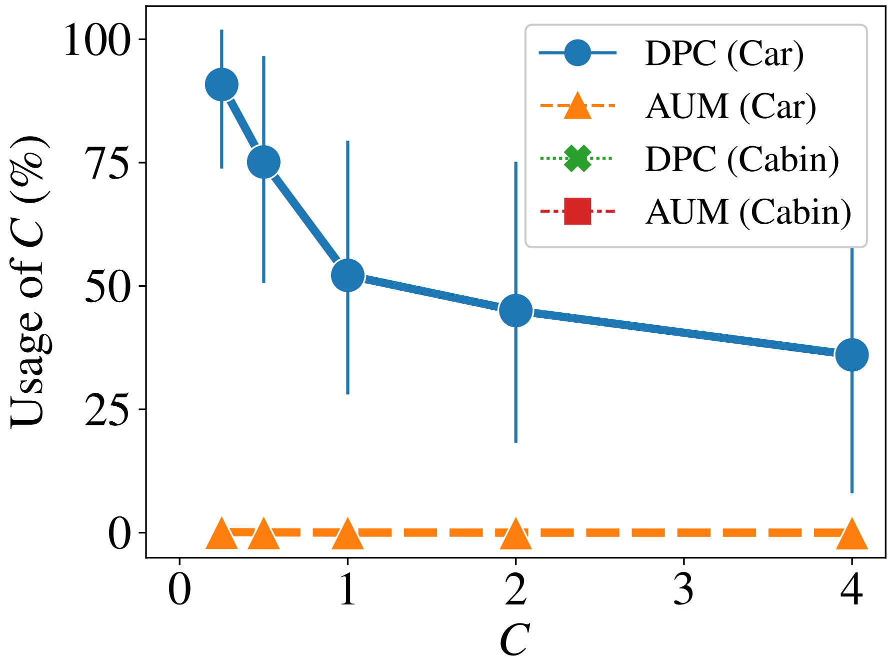


(b)

Figure 6.8: Performance under different C: (a) quality in PSNR and (b) number of input views.

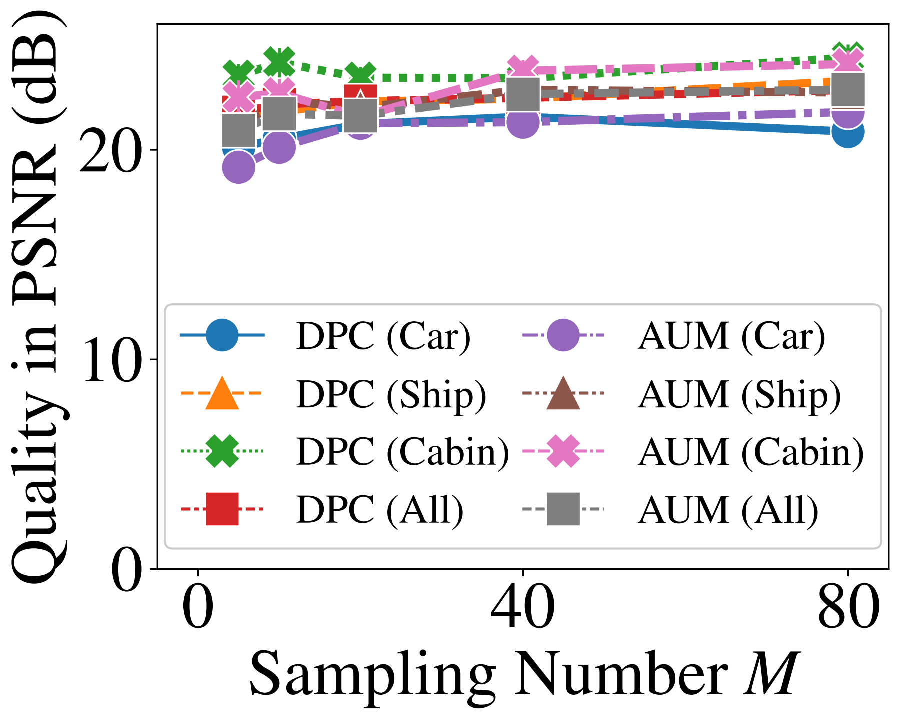


(a)

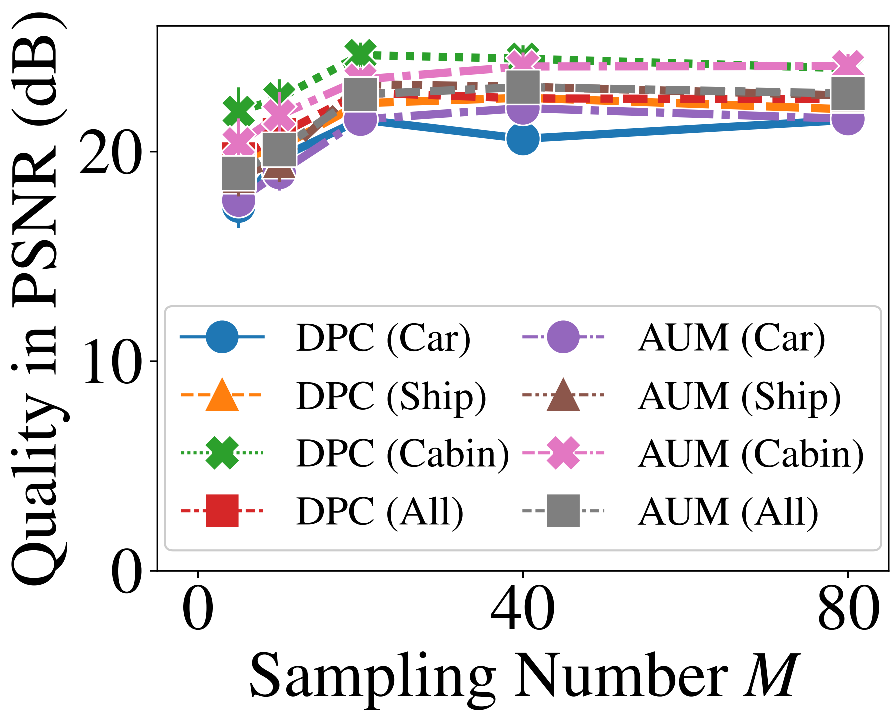


(b)

Figure 6.9: Computational usage under different C: (a) algorithm running time in second and (b) usage.

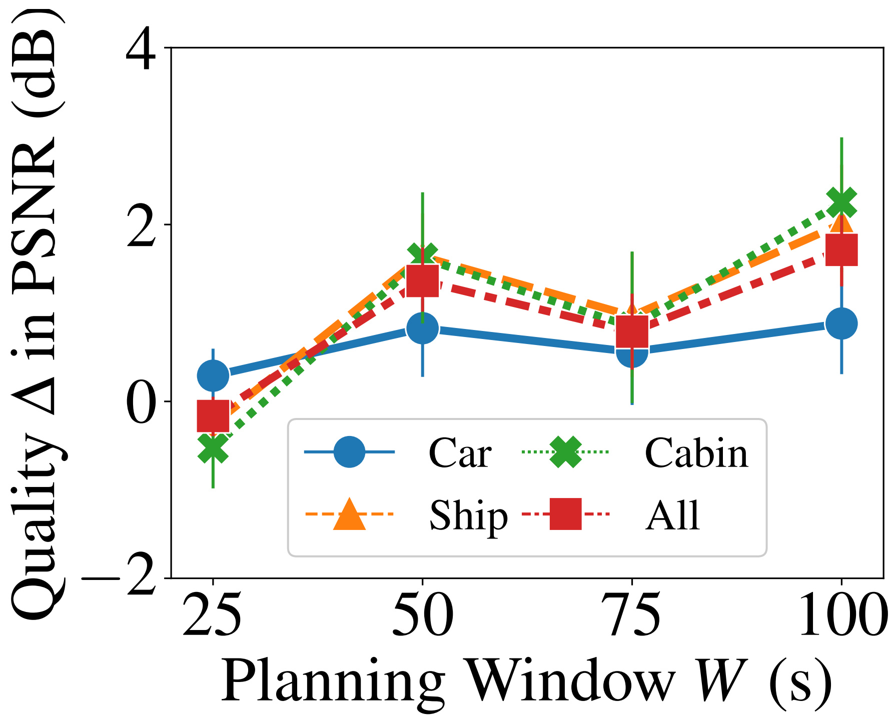


(a)

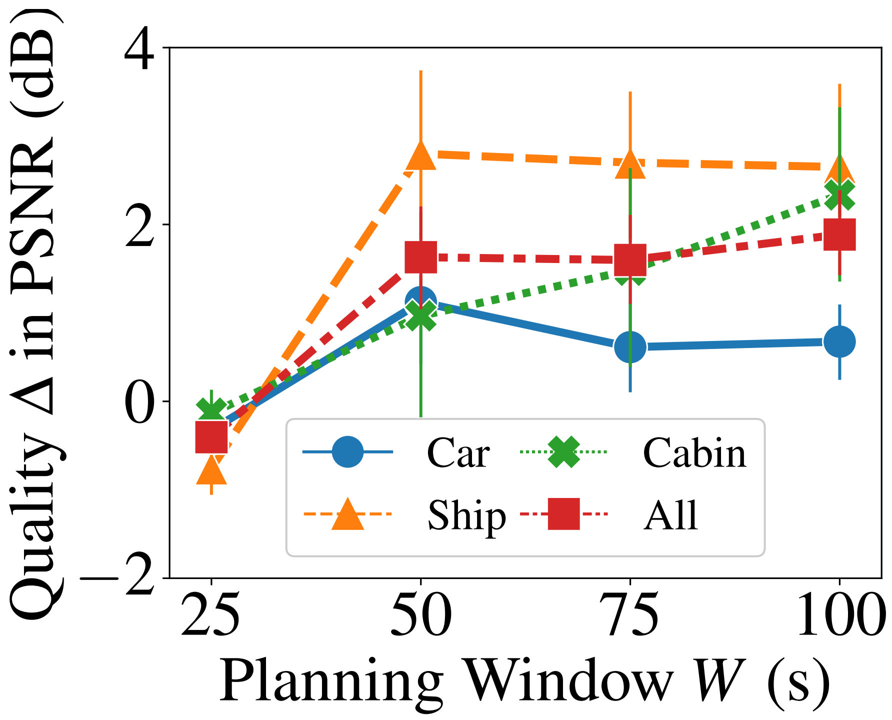


(b)

Figure 6.10: Visual quality under different M from: (a) random and (b) circular sampling.



(a)



(b)

Figure 6.11: Visual quality improvement in PSNR of random sampling over circular sampling under different W using: (a) DPC and (b) AUM.

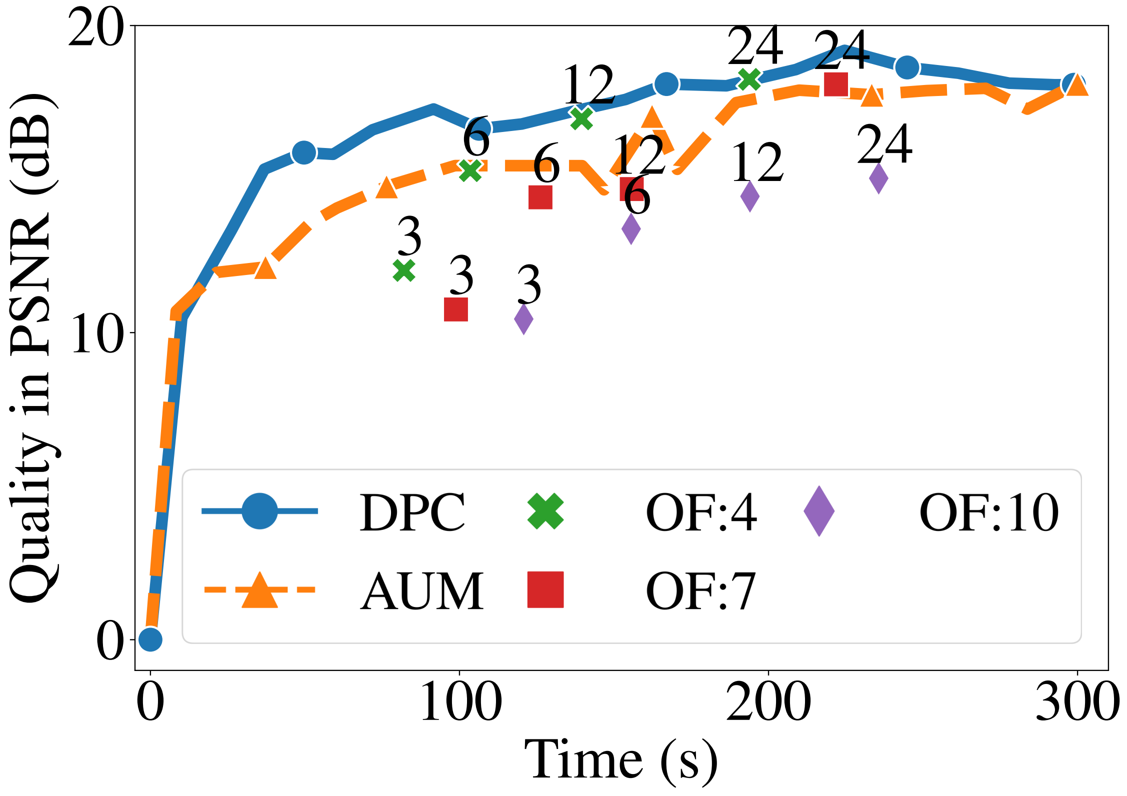
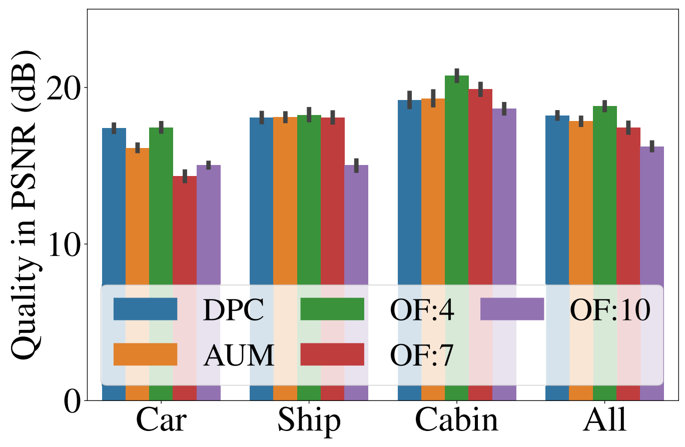
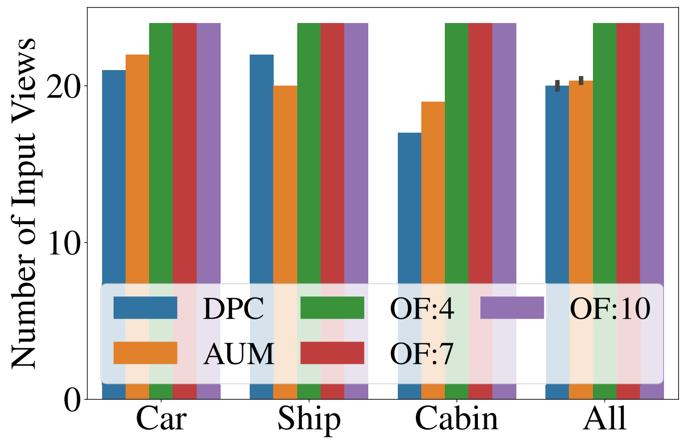


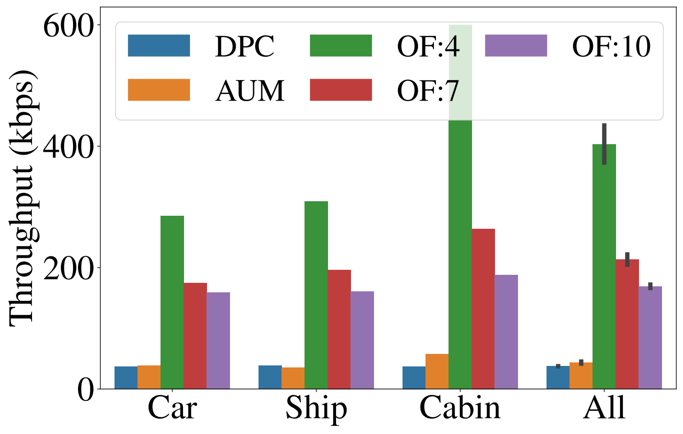
Figure 6.12: Reconstructed visual quality from different algorithms with different radial distances (4, 7, and 10) and numbers of candidate poses (3, 6, 12, and 24).



(a)

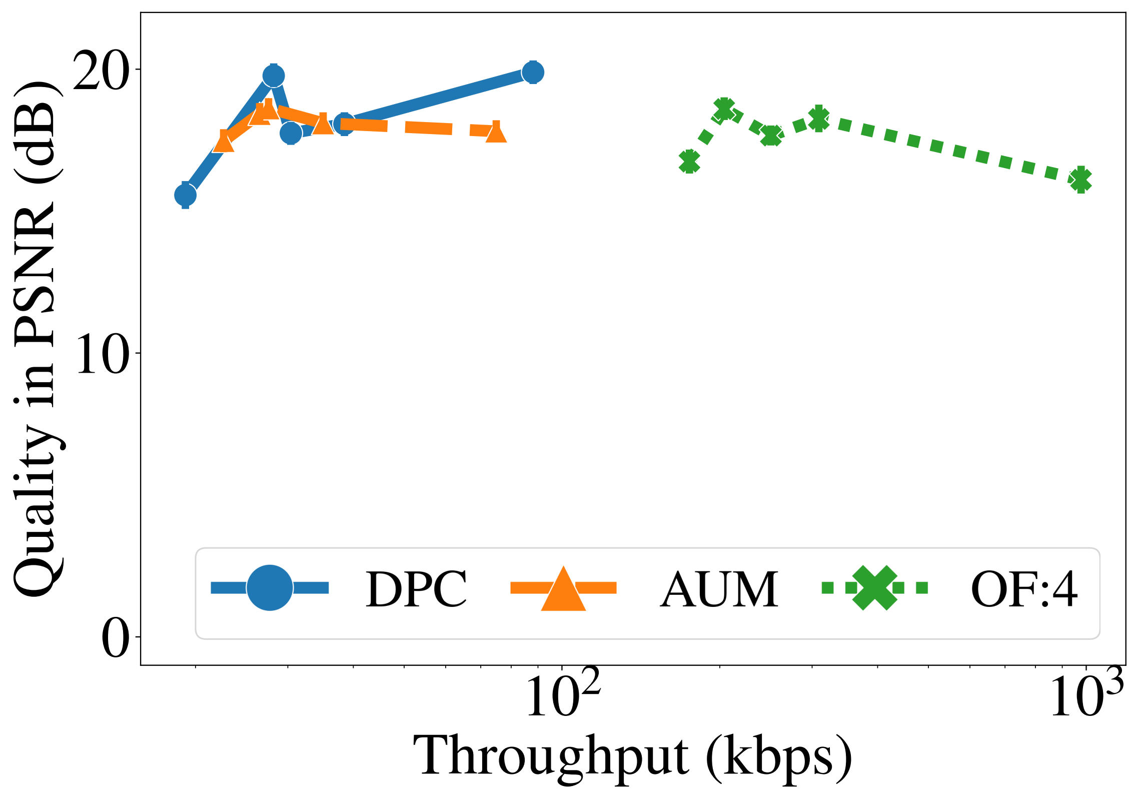


(b)

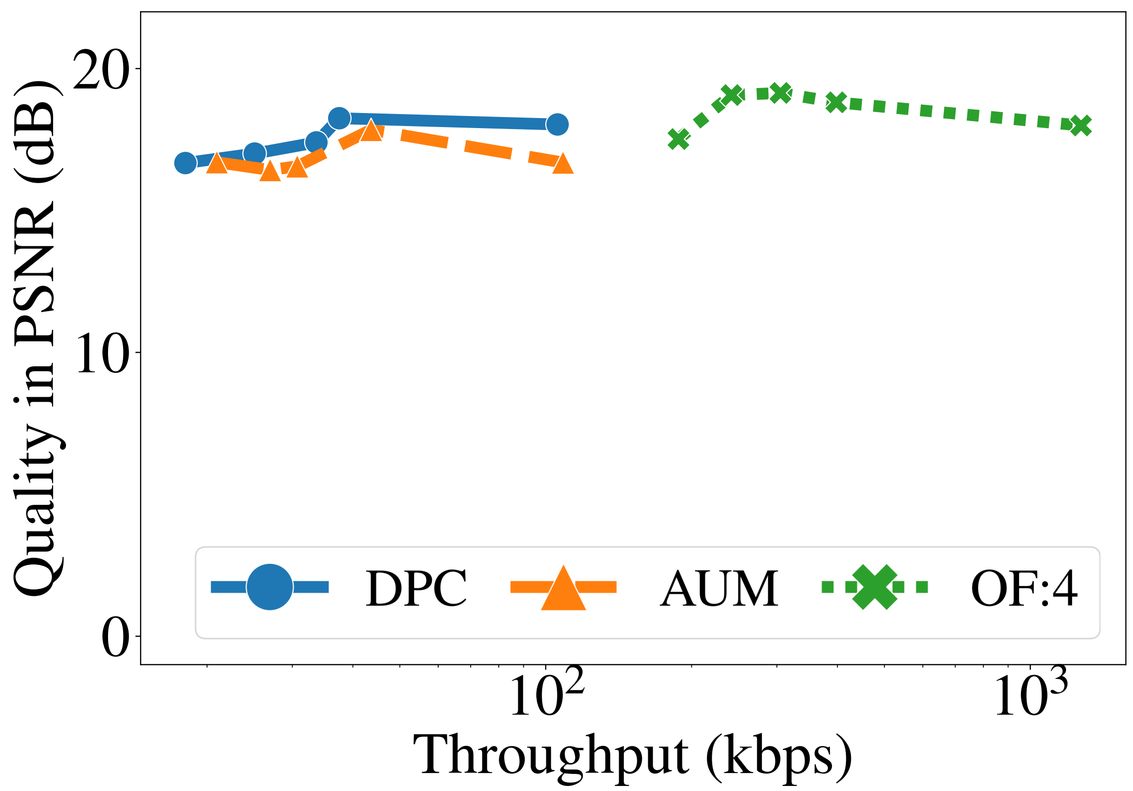


(c)

Figure 6.13: Impacts of different objects: (a) visual quality, (b) number of input images, and (c) uplink throughput.



(a)



(b)

Figure 6.14: Tradeoff between visual quality and uplink throughput from: (a) Ship and (b) average across all three objects.

Chapter 7

Conclusion

7.1 Concluding Remarks

In this study, we designed, implemented, and evaluated a drone-assisted capturing system of 3DGS construction to optimize the resulting synthesized novel views. We used Fisher information [24] to quantify the contributions of individual candidate poses. We then developed two algorithms to solve the NBVS problem for 3DGS objects: (i) the DP-based DPC algorithm for optimal drone trajectories, and (ii) the A*-inspired AUM algorithm for efficient drone trajectories. Experiment results showed that, compared to the prior arts, our solution: (i) improved the visual quality of 3DGS objects by up to 5.90 dB (on average 0.99 dB) in PSNR, (ii) cut the number of input views by up to 48.07% (on average 43.59%), and (iii) achieved the final synthesized view quality of the previous studies in a shorter time (e.g., with a 50+% time reduction for Cabin and random sampling).

7.2 Future Directions

This work can be extended in several directions, including but not limited to: developing NBVS algorithms that better adapt to diverse and dynamic physical environments, enabling multi-drone coordinated capture of 3DGS objects, implementing and evaluating the solution on a real drone, and incorporating more realistic network conditions with packet loss and delay.

- *Reliable protocol designs*: To ensure robust communication between the drone and the ground control station or between drones in a fleet, future work may focus on designing reliable communication protocols tailored to resource-constrained aerial networks. Such protocols should address challenges such as intermittent connectivity, packet loss, and limited bandwidth, while minimizing overhead. Incorporating error correction, adaptive retransmission mechanisms, and QoS (Quality of Service) guarantees can significantly enhance the stability and performance of real-world deployments.

- *Network-Aware Trajectory Planning*: Current trajectory planning focuses primarily on maximizing scene coverage and information gain. A promising direction for future work is to incorporate real-time network metrics—such as signal strength, bandwidth availability, and latency—into the trajectory optimization process. By jointly considering communication quality and NBVS objectives, drones can make more informed decisions that improve both 3DGS reconstruction fidelity and data transmission reliability in dynamic network environments.

- *Uncertainty Estimations*: The current framework adopts Fisher Information [24] as the sole means of quantifying the informational contribution of each view in the 3DGS process. This choice was made due to the limited availability of alternative uncertainty estimation methods at the time of implementation. However, with the rapid development of this field, several novel approaches to uncertainty estimation have recently been proposed [60]. Future research should consider integrating and systematically evaluating these emerging techniques, as they may offer enhanced accuracy in modeling epistemic uncertainty and improved guidance for view selection in online reconstruction scenarios.

- *Cooperation Systems*: Extending the system to support coordinated multi-drone operations would allow for faster and more comprehensive 3DGS scene capture. Future work can explore distributed NBVS algorithms, inter-drone communication strategies, and dynamic task allocation frameworks that enable drones to collaboratively select viewpoints and share reconstruction tasks. Challenges such as collision avoidance, load balancing, and synchronization will need to be addressed for effective system-level cooperation.

- *Alternative Networks*: In addition to traditional Wi-Fi, future implementations could leverage alternative communication technologies such as LoRa (Long Range) or DSRC (Dedicated Short Range Communications). These protocols offer distinct advantages in terms of range, power consumption, and reliability under specific environmental constraints. Investigating their integration into drone-based systems, particularly in remote or infrastructure-limited areas, could broaden the applicability and resilience of 3DGS capture solutions.

Bibliography

[1] Racing car, 2022. https://assetstore.unity.com/packages/3d/vehicles/land/arcade-free-racing-car-161085.

[2] Unity3D game engine. https://unity.com/cn/releases/editor/whats-new/2022.3.7, 2023.

[3] Buildings Module in NS-3. https://www.nsnam.org/docs/models/html/buildings.html, 2025.

[4] ArduPilot Development Team. ArduPilot, 2025. http://www.ardupilot.com.

[5] I. B. Cabin. https://3dwarehouse.sketchup.com/model/1181ab53-5f0c-4c65-8962-deea50abcd51/cabin?hl=en, 2022.

[6] S. Baidya, Z. Shaikh, and M. Levorato. Flynetsim: An open source synchronized uav network simulator based on ns-3 and ardupilot. In *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 37–45, 2018.

[7] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart. Receding horizon" next-best-view" planner for 3D exploration. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 1462–1468. IEEE, 2016.

[8] P. Bouman, N. Agatz, and M. Schmidt. Dynamic programming approaches for the traveling salesman problem with drone. *Networks*, 72(4):528–542, 2018.

[9] L. Chen, S. Peng, and X. Zhou. Towards efficient and photorealistic 3D human reconstruction: a brief survey. *Visual Informatics*, 5(4):11–19, 2021.

[10] R. Clark. Volumetric bundle adjustment for online photorealistic scene capture. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6124–6132, June 2022.

[11] H. Dhami, V. D. Sharma, and P. Tokekar. Map-nbv: Multi-agent prediction-guided next-best-view planning for active 3d object reconstruction. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5724–5731. IEEE, 2024.

[12] S. C. Dhongdi, M. P. Tahiliani, O. Mehta, M. Dharmadhikari, V. Agrawal, and A. Bidwai. Fans: flying ad-hoc network simulator. In *Proceedings of the 2022 Latin America Networking Conference*, pages 34–41, 2022.

[13] F. Duchoň, A. Babinec, M. Kajan, P. Beňo, M. Florek, T. Fico, and L. Jurišica. Path planning with modified A-star algorithm for a mobile robot. *Procedia engineering*, 96:59–69, 2014.

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

[15] C. Forster, M. Pizzoli, and D. Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 15–22. IEEE, 2014.

[16] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart. RotorS—a modular Gazebo MAV simulator framework. *Robot Operating System (ROS) The Complete Reference (Volume 1)*, pages 595–625, 2016.

[17] K. Gao, Y. Gao, H. He, D. Lu, L. Xu, and J. Li. NeRF: Neural radiance field in 3D vision, a comprehensive review. *arXiv preprint arXiv:2210.00379*, 2022.

[18] 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 *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6941–6948. IEEE, 2019.

[19] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun. Deep learning for 3D point clouds: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 43(12):4338–4364, 2020.

[20] P. Hintjens. *ZeroMQ: messaging for many applications*. " O'Reilly Media, Inc.", 2013.

[21] A. Hore and D. Ziou. Image quality metrics: PSNR vs. SSIM. In *2010 20th international conference on pattern recognition*, pages 2366–2369. IEEE, 2010.

[22] C. Hsu, Y.-C. Sun, K.-Y. Lee, and C.-Y. Huang. Will neural 3D object representations be the silver bullet for improving VR experience in HMDs? In *2024 IEEE 7th International Conference on Multimedia Information Processing and Retrieval (MIPR)*, CA, USA, 2024.

[23] A. Y. Javaid, W. Sun, and M. Alam. Uavsim: A simulation testbed for unmanned aerial vehicle network cyber security analysis. In *2013 ieee globecom workshops (gc wkshps)*, pages 1432–1436. IEEE, 2013.

[24] W. Jiang, B. Lei, and K. Daniilidis. FisherRF: Active view selection and uncertainty quantification for radiance fields using Fisher information. *arXiv preprint arXiv:2311.17874*, 2023.

[25] L. Jin, X. Chen, J. Rückin, and M. Popović. NeU-NBV: Next best view planning using uncertainty estimation in image-based neural rendering. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11305–11312. IEEE, 2023.

[26] R. Jin, Y. Gao, Y. Wang, Y. Wu, H. Lu, C. Xu, and F. Gao. Gs-planner: A gaussian-splatting-based planning framework for active high-fidelity reconstruction. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11202–11209, 2024.

[27] Y. Jin, K. Hu, J. Liu, F. Wang, and X. Liu. From capture to display: A survey on volumetric video. *arXiv preprint arXiv:2309.05658*, 2023.

[28] J. M. Jurado, A. López, L. Pádua, and J. J. Sousa. Remote sensing image fusion on 3D scenarios: A review of applications for agriculture and forestry. *International journal of applied earth observation and geoinformation*, 112:102856, 2022.

[29] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis. 3D Gaussian Splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4):1–14, 2023.

[30] C. Koch, K. Georgieva, V. Kasireddy, B. Akinci, and P. Fieguth. A review on computer vision based defect detection and condition assessment of concrete and asphalt civil infrastructure. *Advanced engineering informatics*, 29(2):196–210, 2015.

[31] 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 (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. Ieee, 2004.

[32] A. Koubâa, A. Allouch, M. Alajlan, Y. Javed, A. Belghith, and M. Khalgui. Micro air vehicle link (mavlink) in a nutshell: A survey. *IEEE Access*, 7:87658–87680, 2019.

[33] Y. Li, Z. Kuang, T. Li, G. Zhou, S. Zhang, and Z. Yan. Activesplat: High-fidelity scene reconstruction through active gaussian splatting. *arXiv preprint arXiv:2410.21955*, 2024.

[34] Z. Ma and S. Liu. A review of 3D reconstruction techniques in civil engineering and their applications. *Advanced Engineering Informatics*, 37:163–174, 2018.

[35] M. Maboudi, M. Homaei, S. Song, S. Malihi, M. Saadatseresht, and M. Gerke. A review on viewpoints and path planning for UAV-based 3D reconstruction. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 16:5026–5048, 2023.

[36] MAVLink Development Team. MAVLink common message set. `https://mavlink.io/en/messages/common.html`, 2025. Accessed: 2025-06-01.

[37] L. Meier, D. Honegger, and M. Pollefeys. Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 6235–6240. IEEE, 2015.

[38] L. Meier, P. Tanskanen, L. Heng, G. H. Lee, F. Fraundorfer, and M. Pollefeys. Pixhawk: A micro aerial vehicle design for autonomous flight using onboard computer vision. *Autonomous robots*, 33:21–39, 2012.

[39] Michael Oborne et al. Mission Planner: Ground control station for ardupilot-based uavs. `https://ardupilot.org/planner/`, 2025. Accessed: 2025-06-01.

[40] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[41] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. Orb-slam: A versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

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

[43] X. Pan, Z. Lai, S. Song, and G. Huang. ActiveNeRF: Learning where to see with uncertainty estimation. In *European Conference on Computer Vision*, pages 230–246. Springer, 2022.

[44] QGroundControl Development Team. Qgroundcontrol: Open-source ground control station for drones. `https://qgroundcontrol.com/`, 2025. Accessed: 2025-06-01.

[45] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, et al. ROS: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[46] Y. Ran, J. Zeng, S. He, J. Chen, L. Li, Y. Chen, G. Lee, and Q. Ye. NeurAR: Neural uncertainty for autonomous 3D reconstruction with implicit neural representations. *IEEE Robotics and Automation Letters*, 8(2):1125–1132, 2023.

[47] G. F. Riley and T. R. Henderson. The ns-3 network simulator. In *Modeling and tools for network simulation*, pages 15–34. Springer, 2010.

[48] B. Rodriguez-Garcia, H. Guillen-Sanz, D. Checa, and A. Bustillo. A systematic review of virtual 3D reconstructions of cultural heritage in immersive virtual reality. *Multimedia Tools and Applications*, pages 1–51, 2024.

[49] M. S. M. Sajjadi, H. Meyer, E. Pot, U. Bergmann, K. Greff, N. Radwan, S. Vora, M. Lučić, D. Duckworth, A. Dosovitskiy, J. Uszkoreit, T. Funkhouser, and A. Tagliasacchi. Scene representation transformer: Geometry-free novel view synthesis through set-latent scene representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6229–6238, June 2022.

[50] L. Savant, D. Valsesia, and E. Magli. Modeling uncertainty for Gaussian Splatting. *arXiv preprint arXiv:2403.18476*, 2024.

[51] 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*, pages 621–635. Springer, 2018.

[52] J. Shen, A. Agudo, F. Moreno-Noguer, and A. Ruiz. Conditional-Flow NeRF: Accurate 3D modelling with reliable uncertainty quantification. In *European Conference on Computer Vision*, pages 540–557. Springer, 2022.

[53] S. Song, D. Kim, and S. Choi. View path planning via online multiview stereo for 3D modeling of large-scale structures. *IEEE Transactions on Robotics*, 38(1):372–390, 2021.

[54] S. T. Ship. `https://assetstore.unity.com/packages/3d/environments/flooded-grounds-48529#description`, 2019.

[55] S.-M. Tang, C.-H. Hsu, Z. Tian, and X. Su. An aerodynamic, computer vision, and network simulator for networked drone applications. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, pages 831–833, 2021.

[56] S.-M. Tang, Y.-C. Sun, and C.-H. Hsu. A blind streaming system for multi-client online 6-DoF view touring. In *Proceedings of the 31st ACM International Conference on Multimedia*, pages 9124–9133, 2023.

[57] A. Varga. Omnet++. In *Modeling and tools for network simulation*, pages 35–59. Springer, 2010.

[58] S. Verykokou, A. Doulamis, G. Athanasiou, C. Ioannidis, and A. Amditis. UAV-based 3D modelling of disaster scenes for urban search and rescue. In *2016 IEEE International Conference on Imaging Systems and Techniques (IST)*, pages 106–111, 2016.

[59] N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang. Pixel2Mesh: Generating 3D mesh models from single RGB images. In *Proceedings of the European conference on computer vision (ECCV)*, pages 52–67, 2018.

[60] J. Wilson, M. Almeida, S. Mahajan, M. Labrie, M. Ghaffari, O. Ghasemalizadeh, M. Sun, C.-H. Kuo, and A. Sen. Pop-gs: Next best view in 3d-gaussian splatting with p-optimality. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 3646–3655, 2025.

[61] Z. Xu, R. Jin, K. Wu, Y. Zhao, Z. Zhang, J. Zhao, F. Gao, Z. Gan, and W. Ding. Hgs-planner: Hierarchical planning framework for active scene reconstruction using 3d gaussian splatting. *arXiv preprint arXiv:2409.17624*, 2024.

[62] X. Yu and C. W. Chen. Semantic-aware next-best-view for multi-dofs mobile system in search-and-acquisition based visual perception. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 3713–3721, 2024.

[63] J. Zeng, Y. Li, Y. Ran, S. Li, F. Gao, L. Li, S. He, J. Chen, and Q. Ye. Efficient view path planning for autonomous implicit reconstruction. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4063–4069. IEEE, 2023.

[64] J. Zeng, Y. Li, J. Sun, Q. Ye, Y. Ran, and J. Chen. Autonomous implicit indoor scene reconstruction with frontier exploration. *arXiv preprint arXiv:2404.10218*, 2024.

[65] J. Zeng, Q. Ye, T. Liu, Y. Xu, J. Li, J. Xu, L. Li, and J. Chen. Multi-robot autonomous 3d reconstruction using gaussian splatting with semantic guidance. *IEEE Robotics and Automation Letters*, 2025.