TU WIEN Informatics

Using a drone for automated 3D Scanning

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Klemens Johannes Wiesinger
Matrikelnummer 11938253

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Projektass. Stefan Ohrhallinger, Mag.rer.soc.oec. PhD

Wien, 26. März 2025

Klemens Johannes Wiesinger	Michael Wimmer

TU WIEN Informatics

Using a drone for automated 3D Scanning

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Klemens Johannes Wiesinger
Registration Number 11938253

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Projektass. Stefan Ohrhallinger, Mag.rer.soc.oec. PhD

Vienna, March 26, 2025

Klemens Johannes Wiesinger	Michael Wimmer

Erklärung zur Verfassung der Arbeit

Klemens Johannes Wiesinger

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 26. März 2025

Klemens Johannes Wiesinger

Danksagung

An erster Stelle möchte ich mich bei meinen Eltern bedanken – bei meinem Vater für seine handwerkliche Unterstützung bei den praktischen Teilen des Projekts, bei meiner Mutter für die nächtlichen Telefonate, die mir durch die schwierigeren Phasen geholfen haben, und für alles andere, was die beiden für mich im Laufe des Studiums getan haben.

Ein herzliches Dankeschön geht an alle am Institut für Computergraphik für eure Unterstützung und dafür, dass ihr während der Testphase den Lärm ertragen habt. Besonderer Dank gilt Annalena und Markus, die immer ein offenes Ohr hatten, wenn ich meinen Frust über das Projekt loswerden musste.

Besonders dankbar bin ich meinem Betreuer Stefan Ohrhallinger dafür, dass er mir ermöglicht hat, diese Arbeit in einem engen Zeitrahmen abzuschließen.

Auch wenn sie nicht direkt an dieser Arbeit beteiligt waren, möchte ich Ulf Assarsson und Erik Sintorn herzlich danken – sie haben mir gezeigt, wie inspirierend und angenehm ein Forschungsumfeld sein kann. Eure Mentorschaft hat einen bleibenden Eindruck hinterlassen. Ein weiteres großes Dankeschön geht an meinen Mentor Michael Wimmer für seine Unterstützung während meines Bachelor with Honors-Studiums.

An all meine Freunde: Danke, dass ihr mich in den letzten Monaten (größtenteils) bei Verstand gehalten habt. Eure Unterstützung, euer Zuspruch und eure Ablenkung kamen immer genau zur richtigen Zeit.

Zu guter Letzt: Danke an alle, die mir auf diesem Weg geholfen haben – sei es durch Ratschläge, Motivation oder einfach durch ihre Anwesenheit. Ohne euch hätte ich das nicht geschafft.

Acknowledgements

First and foremost, I want to thank my parents—my father for lending his hands-on skills to the practical parts of the project, and my mother for the late-night calls that got me through the tougher stages, and for everything else they've done for me throughout my studies.

A heartfelt thank you goes out to everyone at the Institute of Computer Graphics for your support and for putting up with the noise during the testing phase. A special thanks to Annalena and Markus, who were always there to listen whenever I needed to vent about the project.

I'm especially grateful to my supervisor, Stefan Ohrhallinger, for making it possible to complete this thesis within a tight timeframe.

Although not directly involved in this thesis, I owe special thanks to Ulf Assarsson and Erik Sintorn for showing me how inspiring and enjoyable a research environment can be. Your mentorship left a lasting impression. I'm also deeply grateful to my mentor, Michael Wimmer, for supporting me throughout my Bachelor with Honors journey.

To all my friends: thank you for keeping me (mostly) sane throughout these past months. Your support, encouragement, and perfectly timed distractions made all the difference.

And finally, to everyone who helped me along the way—whether through advice, motivation, or simply being there—thank you. I couldn't have done this without you.

Kurzfassung

Drohnen haben in den letzten Jahrzehnten in diversen Bereichen Anwendung gefunden und unterstützen kritische Aufgaben wie Such- und Rettungseinsätze, Inspektionen und Vermessungen. Autonomes 3D-Scannen mit Drohnen könnte solche Einsätze weiter verbessern, stellt jedoch insbesondere in Innenräumen eine große Herausforderung dar. Diese Arbeit untersucht die Machbarkeit des autonomen 3D-Scannens in Innenräumen mit einer handelsüblichen Drohne. Unser Ansatz kombiniert eine DJI Spark Drohne mit einem leichten Pico Flexx Tiefensensor und einem Raspberry Pi zur Erfassung von Tiefendaten, die in Echtzeit an einen externen Server übertragen werden. Das System basiert auf ROS 2 und InfiniTAM zur simultanen Lokalisation und Kartierung (SLAM) sowie zur 3D-Rekonstruktion. Die Navigationsbefehle werden über ein Smartphone mit Hilfe von DJIs Mobile SDK an die Drohne übermittelt.

Obwohl ein begrenzter autonomer Scan erfolgreich durchgeführt wurde, traten zahlreiche Einschränkungen auf — darunter die geringe Traglast der Drohne, die eingeschränkte Reichweite des Sensors sowie Hardwareinstabilitäten. Trotz dieser Hürden wurde eine modulare Softwarearchitektur entwickelt, die Sensordatenerfassung, Kartierung und Navigation integriert. Diese bildet eine solide Grundlage für zukünftige Entwicklungen in Richtung vollständig autonomer 3D-Scans in Innenräumen mit leistungsfähigerer Hardware. Die Generierung des nächsten besten Blickwinkels und die Pfadfindung dorthin bleiben jedoch offene Herausforderungen.

Abstract

Drones have been widely adopted over the past decades, aiding in critical tasks such as search and rescue, inspection, and mapping. Autonomous drone scanning could further support such missions, but remains a significant challenge in indoor environments. This thesis explores the feasibility of using a consumer drone for such autonomous indoor 3D scanning. Our approach combines a DJI Spark drone, a lightweight Pico Flexx depth sensor, and a Raspberry Pi to capture depth data, which is streamed to an external server for real-time processing. The system leverages ROS 2 and InfiniTAM for SLAM and map reconstruction, while navigation commands are issued via a smartphone using DJI's Mobile SDK.

Although the system successfully completed a limited autonomous scan, various constraints—including the drone's payload capacity, limited sensor range, and hardware instabilities—posed significant challenges. Despite these limitations, a modular software architecture was developed that integrates sensing, mapping, and navigation. This framework provides a solid foundation for future work toward fully autonomous indoor scanning with more capable hardware. However, generating the next best view and finding a feasible path toward it remain open challenges.

Contents

Kurzfassung **xi**

Abstract **xiii**

Contents **xv**

1 Introduction **1**

2 Related Work **3**
2.1 Simultaneous Location and Mapping - SLAM 3
2.2 Space Exploration for Visual SLAM 4
2.3 Path Planning in Unknown Environments 5
2.4 Middleware for Robotic Systems – ROS and ROS2 7
2.5 Overview of Drone-Based Indoor Scanning and Mapping Projects . . . 7

3 Method **9**
3.1 Planned System Architecture 9
3.2 Implemented System 10
3.3 Implementation Limitations 14

4 Evaluation **17**
4.1 Mounting the Raspberry Pi 17
4.2 Drone stability 20
4.3 Scan Process Evaluation 22
4.4 System Comparison 25

5 Conclusion **27**
5.1 Future Work 27

Overview of Generative AI Tools Used **29**

List of Figures **31**

List of Tables **33**

List of Algorithms **35**

Bibliography **37**

CHAPTER 1

Introduction

Over the past decades, drones have become increasingly common across various domains. In civil applications, they are used for tasks such as architectural documentation, infrastructure inspection in civil engineering, and precision agriculture for crop monitoring and fertilization. Drones have also been used in more critical applications such as disaster response and monitoring, including Search and Rescue (SAR) missions [ENCA17, AR24].

Most of these uses still rely on manual control, especially for indoor operations. In outdoor environments, drones can navigate using GPS and predefined waypoints, often avoiding obstacles simply by flying at higher altitudes. Indoors, however, GPS is unavailable, and the environment is typically more cluttered and constrained, making autonomous navigation significantly more complex.

Autonomous exploration and scanning could greatly assist in critical missions such as SAR, potentially significantly increasing response speed and safety. By eliminating the need for direct human control, multiple drones can be deployed in parallel, covering more ground in less time and reducing the burden on rescue personnel.

Objective and Methodology

With this thesis, we aim to take a step toward realizing that vision by evaluating the feasibility of using a low-cost consumer drone for fully autonomous indoor 3D scanning and proposing a modular system architecture to support this process. A complete system should be capable of autonomously exploring an indoor environment by computing the next best position to scan from, navigating to that pose, and avoiding obstacles along the way. While a full implementation of such a system is beyond the scope of this project, our work aims to make progress toward this goal.

To this end, we built a prototype system around the DJI Spark drone, augmented with a lightweight Pico Flexx depth sensor. To keep the drone as light as possible, the sensor

data is streamed wirelessly to an external server for processing. This server performs Simultaneous Localization and Mapping (SLAM) and generates flight commands for the drone. The software stack is built on ROS 2 [MFG+22] and uses InfiniTAM [KPR+15, KPM16, PKG+17], for visual SLAM, allowing the system to reconstruct a 3D environment model incrementally.

The remainder of this thesis is organized as follows. Chapter 2 reviews existing work on SLAM, autonomous exploration, path planning, and related drone scanning systems. Chapter 3 describes the system design and implementation, including both the planned and realized architecture. Chapter 4 evaluates the system regarding drone stability, scanning performance, and practical limitations. Finally, Chapter 5 summarizes the findings and outlines directions for future work.

CHAPTER 2

Related Work

This chapter outlines the foundations and existing approaches relevant to our system. We begin with an overview of SLAM methods before exploring strategies for spatial exploration and path planning in unknown environments. We then introduce ROS(2), a widely used middleware in robotics, and conclude with a survey of drone-based indoor scanning projects related to our work.

2.1 Simultaneous Location and Mapping - SLAM

This section is based on the state-of-the-art report by Kazerouni et al. [AFDT22], which provides a comprehensive overview of SLAM methods, sensor technologies, and prominent Visual SLAM systems.

Simultaneous Localization and Mapping (SLAM) is a foundational concept that enables a system to build a map of an unknown environment while simultaneously determining its own position within that map, and it is widely used in fields such as robotics, augmented reality, and computer vision modeling. This ability is essential for autonomous operation in GPS-denied settings, such as indoor environments. Rather than being a single algorithm, SLAM refers to a family of methods that combine perception, motion estimation, and mapping into a unified framework.

2.1.1 Visual SLAM

Visual SLAM (V-SLAM) is a subset of SLAM methods that rely primarily on visual sensors—such as cameras—for both mapping and localization. Unlike traditional SLAM systems that may use LiDAR or sonar, V-SLAM systems process image data to infer motion and structure. Visual information is used to track motion (through visual odometry), recognize previously visited locations (loop closure), and incrementally build a map of the environment. V-SLAM has become particularly important in scenarios

where lightweight and low-cost sensors are preferred, such as in mobile robotics and AR/VR.

There are two broad categories of methods in Visual SLAM: feature-based (indirect) and direct approaches. Feature-based methods extract and match distinct visual keypoints (e.g., corners or edges) across frames. A leading example is ORB-SLAM3 [CER+21], which supports monocular, stereo, and RGB-D input and is known for its robustness and versatility. Direct methods, on the other hand, operate on raw image intensities without relying on discrete features. Examples include LSD-SLAM [ESC14] and DSO [EKC16], which are particularly useful in low-texture environments.

Dense reconstruction methods within V-SLAM utilizing RGB-D cameras have become prevalent. Prominent examples include KinectFusion [NIH+11], which is depth-only, and ElasticFusion [WLSM+15], which utilizes full RGB-D data. KinectFusion, although relying exclusively on depth data, laid the foundation for many subsequent RGB-D SLAM approaches and is commonly categorized under Visual SLAM. These dense reconstruction methods are particularly valuable for applications such as indoor scanning and augmented reality.

V-SLAM can be implemented using a variety of cameras, including monocular, stereo, and RGB-D setups. The choice of sensor affects not only the accuracy and completeness of the map but also the system's ability to infer scale and depth. More recently, systems have begun to incorporate data from inertial measurement units (IMUs) and other sensors for improved robustness and drift reduction.

2.2 Space Exploration for Visual SLAM

To obtain a complete scan of a room, the robot must explore all observable space. This requires planning where to move next in order to reveal unknown areas or reduce map uncertainty. The most common strategies fall into three categories: frontier-based exploration, Next-Best-View (NBV) planning, and hybrid approaches that combine elements of both.

2.2.1 Frontier-Based Exploration

Frontier-based exploration was first introduced by Yamauchi [Yam97] in 1997 as a strategy for guiding robots toward the boundary between known free space and unexplored regions—so-called "frontiers"—to incrementally expand their map of the environment. More recently, Batinovic et al.[BPI+21] proposed a multi-resolution planner that clusters frontiers to support UAV navigation in large-scale 3D environments.

2.2.2 Next-Best-View Planning

Next-Best-View (NBV) planning selects the next sensor pose to maximize the expected information gain. It is often framed as an information gain maximization problem, guiding

the robot toward views that reduce map uncertainty or improve coverage. Approaches vary in how they formulate this objective—from local sampling-based methods to continuous trajectory optimization. For instance, Zhao et al.[ZXZ+22] introduced a receding horizon NBV strategy for MAVs that integrates loop closure and obstacle avoidance, while Haner and Heyden [HH11] proposed an optimization framework that balances uncertainty reduction with path length.

In addition to classical planning strategies, learning-based methods such as reinforcement learning and supervised view prediction have been explored [WXC+24][MVGT+20]. These approaches typically focus on object-centric tasks and remain limited in their applicability to room-scale scanning.

2.2.3 Hybrid Approaches

Hybrid approaches have emerged to address the limitations of using frontier- or NBV-based strategies in isolation. Lu et al.[LDLMT22] proposed an Optimal Frontier Enhanced NBV Planner (OFENBVP) that flexibly switches between local NBV planning and global frontier-based guidance. This hybrid method helps robots escape local minima and improves overall exploration efficiency. Similarly, Selin et al.[STD+19] introduced the Autonomous Exploration Planner (AEP), which combines Receding Horizon Next-Best-View Planning (RH-NBVP) locally with a frontier-inspired global planner using cached high-information-gain regions.

2.3 Path Planning in Unknown Environments

Navigating a drone toward a predefined goal in an unknown or partially known environment is a fundamental challenge in robotics. The drone must avoid obstacles and generate safe, feasible trajectories using only onboard sensing and any maps it builds along the way. Many approaches address this problem, most falling into two main categories: A*-based methods, which perform graph search over a discretized map, and RRT-based methods, which sample paths through continuous space.

2.3.1 A*-Based Planning

A*-based methods rely on an explicit map—either known in advance or built incrementally—to find the optimal path from start to goal over a discretized representation of the space. D* (Dynamic A*) [Ste94] is one of the most well-known extensions. It efficiently replans paths as new obstacle information becomes available. This makes it particularly effective in unknown terrain where the map is updated continuously.

Other notable A*-based systems include Hybrid A* [DTMD08], which extends A* into continuous space by planning over discretized poses and smoothing the resulting paths. ORRT-A* [AMS+19] combines A* with sampling by using A* to guide RRT sampling, improving planning performance in partially known or cluttered spaces.

These methods excel in structured or mapped environments, offering optimal paths with high efficiency, but may struggle with continuous or high-dimensional spaces.

2.3.2 RRT-Based Planning

Sampling-based methods such as RRT (Rapidly-exploring Random Tree) and its optimal variant RRT* are popular for high-dimensional and continuous planning problems. These methods grow trees through free space by randomly sampling points and connecting them, making them highly flexible in complex environments.

MOD-RRT* [QYS21] enhances RRT* with multi-objective planning and dynamic replanning capabilities, enabling the system to adapt in real time to newly discovered obstacles. Neural RRT* [WCL+20] further improves convergence speed by training a convolutional neural network to predict likely path distributions based on previous A*-generated paths, which biases the sampling process toward more promising regions.

RRT-based approaches are flexible and well-suited to high-dimensional or unstructured spaces, but can produce suboptimal paths and require post-processing or enhancements to improve efficiency.

2.3.3 Global and Local Planning Architectures

Many systems use a two-level architecture with a global planner for long-term pathfinding and a local planner for short-term, reactive control. This is especially useful in unknown environments, where global plans must be frequently updated as new space is discovered.

For example, DWA-3D [BDRM24] pairs a global RRT* planner with a local dynamic window approach that selects safe, goal-directed velocity commands. Systems using model predictive control (MPC) similarly compute short-horizon, dynamically feasible trajectories to track the global path [KHDK22].

This architecture balances global goal pursuit with local reactivity, improving robustness, though it requires careful coordination between layers.

2.3.4 Deep Reinforcement Learning-Based Planning

Some approaches avoid explicit planning altogether and instead train a neural network to directly map sensory input to control commands using deep reinforcement learning (DRL) [XMW+21]. These methods do not rely on a persistent map or path representation. Instead, they take in local sensor observations (e.g., 3D obstacle grids) and output the next action.

DRL methods are highly reactive and do not require a map, but are harder to train and less effective when structured map data is available.

2.4 Middleware for Robotic Systems – ROS and ROS2

Robotic systems are inherently modular and distributed, typically composed of independent components for sensing, planning, control, and actuation. To enable communication between these modules, robotics frameworks—commonly referred to as *middleware*—are used to structure data exchange and process coordination. One of the most widely adopted frameworks in both research and industry is the Robot Operating System (ROS).

ROS 1, originally described in [QCG+09], became a cornerstone of robotics development, offering a large ecosystem of community-developed packages and tools. However, as the field matured, ROS 1's limitations became more apparent—especially its lack of security, poor support for real-time execution, and reliance on a centralized communication model. These limitations made it unsuitable for many production-grade applications.

ROS 2, a complete redesign of the original framework, addresses these challenges and introduces features essential for modern robotics. As described by Macenski et al. [MFG+22], ROS 2 builds on the *Data Distribution Service* (DDS), an open communication standard widely used in high-reliability domains such as aerospace and finance. DDS enables features such as distributed peer-to-peer communication, quality-of-service (QoS) configuration, and secure, real-time operation in unreliable networks.

The ROS 2 architecture is centered around the following key abstractions:

- **Nodes:** Independent computational entities representing a unit of functionality, such as a sensor driver or planner.
- **Topics:** Asynchronous publish-subscribe communication channels for streaming sensor or state data.
- **Services:** Synchronous request-response interfaces for short-lived tasks or queries.
- **Actions:** Goal-oriented, long-running asynchronous interfaces that support progress feedback and cancellation—commonly used in navigation and manipulation.
- **Parameters:** Runtime-configurable values for tuning node behavior.

These components are organized into a computational graph, facilitating modular design and introspection. ROS 2 introduces additional features such as *lifecycle nodes*, which allow better state management (e.g., activating, deactivating, or resetting modules), and *intra-process communication* for reduced latency when nodes run within the same process.

2.5 Overview of Drone-Based Indoor Scanning and Mapping Projects

Various drone-based systems have been developed for indoor mapping and navigation, differing in sensor configurations, SLAM techniques, and degrees of autonomy. This

section highlights representative approaches that use drones in similar contexts as our project.

Alexovič et al. [ALB23] present a quadrotor drone equipped with an Intel RealSense D435i RGB-D camera for 3D sensing and a T265 tracking camera for visual-inertial odometry. The system uses the RTAB-Map framework for visual SLAM and supports partial autonomy via the ROS Move Base stack. Navigation goals can be issued in a pre-built 2D map, created during manual flight, that reflects the environment at the drone's flying height. The drone localizes itself within this map using the tracking camera and follows planned 2D paths to goal positions using PX4 firmware. While primarily manually operated, this setup allows waypoint-based movement through a ROS2-integrated architecture.

Iyer et al. [ARN+24] describe a fully autonomous navigation framework implemented in simulation using Gazebo and ROS2. The drone uses LiDAR for obstacle detection and mapping and builds a 2D occupancy map using SLAM Toolbox. Navigation is handled via the Nav2 stack, which was adapted to control the drone's movement. The drone operates at a fixed height, effectively constraining motion to a 2D plane.

Krul et al. [KPFV21] evaluate two monocular visual SLAM algorithms, LSD-SLAM and ORB-SLAM, for indoor drone mapping in agricultural environments. Their experimental platform is a DJI Tello, a lightweight consumer drone equipped with a forward-facing monocular camera. The drone is manually controlled during data collection, with SLAM processing and point cloud generation performed externally. In controlled experiments, they define waypoint missions to assess localization stability, demonstrating that SLAM-derived pose estimates can support repeatable waypoint flight. OctoMap is used to build sparse 3D obstacle maps.

Zhang et al. [ZYZ24] present the ICON drone, a fully autonomous indoor UAV system for semantic 3D reconstruction. The drone uses a stereo RGB-D camera and IMU for visual-inertial odometry (VIO), enabling localization in GPS-denied environments. It performs real-time frontier-based exploration and generates smooth, dynamically feasible B-spline trajectories to cover the unknown environment. RGB images collected during flight are used for dense 3D reconstruction, with absolute scale recovered via VIO. Unlike prior methods, the ICON drone system operates without any prior map and autonomously explores and reconstructs indoor scenes.

While drones have been widely used in combination with SLAM for indoor environments, the degree of autonomy in most systems remains limited. Several approaches rely on pre-mapped environments, are constrained to 2D navigation, or follow pre-defined waypoint missions. Zhang et al.'s ICON drone marks a significant step forward by enabling full 3D autonomous exploration and reconstruction without prior maps. Our approach shares similar goals but differs in architecture: rather than relying on onboard computing, we offload all processing to a server and communicate with the drone via Wi-Fi.

CHAPTER 3

Method

This chapter describes both the intended and realized design of our autonomous indoor 3D scanning system. The initial goal was to develop a fully autonomous drone to explore and reconstruct indoor environments using onboard sensors, mapping, and planning algorithms. However, due to hardware constraints and development complexity, only a subset of the full system was implemented. The following sections first outline the ideal architecture before describing the concrete implementation used during the evaluation.

3.1 Planned System Architecture

The planned system architecture is illustrated in Figure 3.1. The architecture separates the responsibilities of the drone and the server. The drone is responsible for capturing sensor data and executing low-level flight commands, while all computation-heavy tasks are performed on an external server, preferably located within the same room to reduce latency.

Sensor data from the onboard sensors is streamed to the server, where it is processed by a (Visual) SLAM system to estimate the drone's pose and build a 3D map in real-time. The resulting point cloud or mesh is passed to the Next-Best-View (NBV) module, which analyzes the current map and selects an optimal viewpoint to extend coverage.

The online path planning module takes the generated NBV target and occupancy data from the SLAM module to compute a feasible flight path from the current position to the target view. This process must run continuously to adapt the path dynamically in case newly discovered obstacles block the originally planned trajectory.

The computed path is then passed to the Drone Command Generation module, which takes into account the drone's current pose to generate appropriate low-level control commands (e.g., velocity or position targets) that guide the drone along the planned path. These commands are then sent back to the drone for execution.

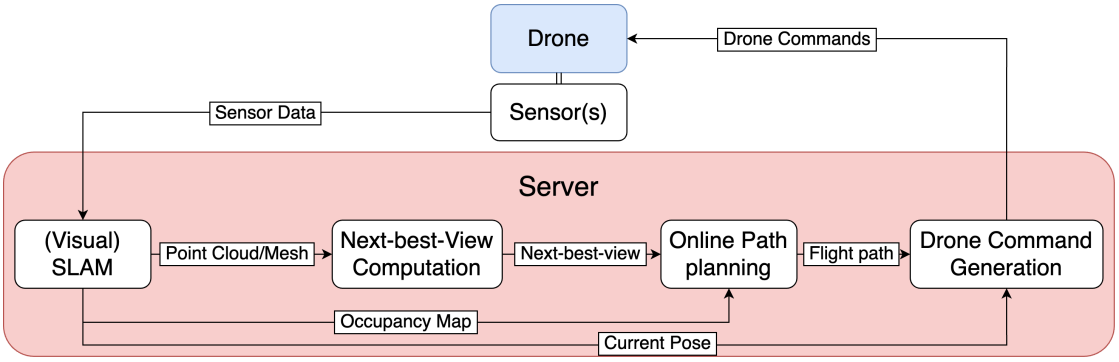


Figure 3.1: Planned system architecture for autonomous indoor 3D scanning.

3.2 Implemented System

Having presented an overview of the planned system architecture, we now turn to the system as it was actually implemented. An overview of the components and communication pathways in the final setup is shown in Figure 3.2.

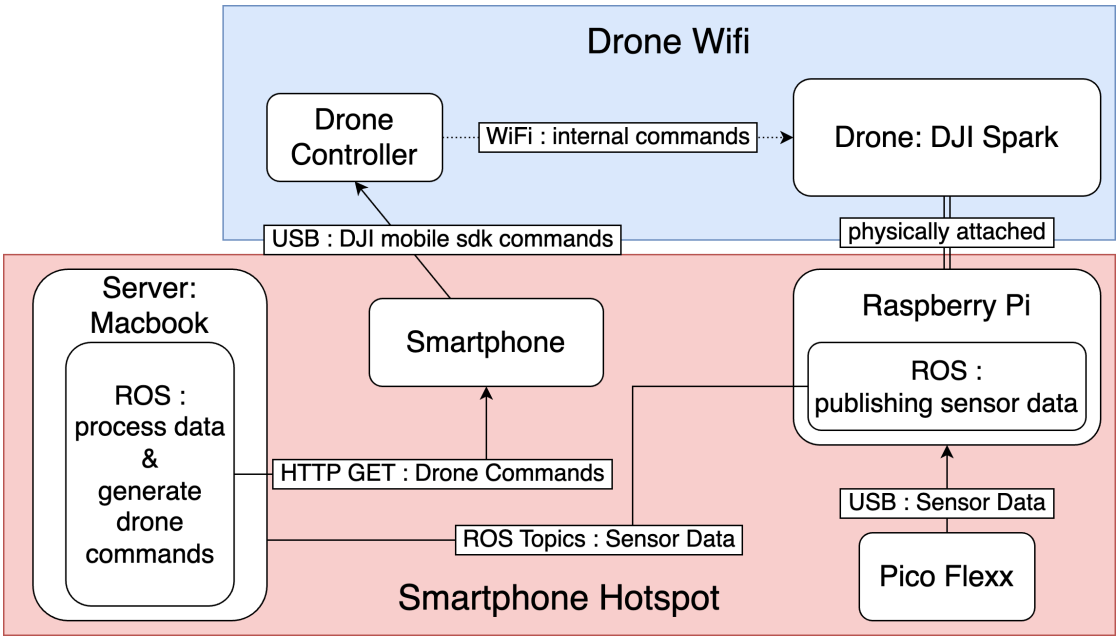


Figure 3.2: Overview of the implemented system. Sensor data is published via ROS on the Raspberry Pi and transmitted over the smartphone's hotspot network to the server. The server processes the data and sends flight commands to the smartphone via HTTP GET requests, which relays them to the drones controller via USB.

3.2.1 Hardware Setup

To keep costs low, we used components that were either already available or inexpensive to acquire. The core of our setup was a DJI Spark drone, which carried a Pico Flexx time-of-flight sensor to capture depth data for 3D reconstruction. The sensor was connected to a Raspberry Pi Zero 2 W, which streamed the data over a local network to a laptop running Ubuntu 22.04, acting as our processing server.

Since the Spark can only be controlled using DJI's mobile SDK, a Google Pixel 8 smartphone acted as a relay, forwarding control commands from the server to the drone. The phone also hosted a mobile hotspot, forming an isolated network between the Pi, the server, and the phone itself.

Since both the Spark and its controller emit Wi-Fi signals that only support a single connected device at a time, we could not use these networks to transmit all data within the system. At the same time, the phone needed to be on the same network as the server to receive the generated commands. The only solution was to have the phone connected to the same network as the server while simultaneously connecting it to the drone's remote controller via a USB cable to forward commands.

A photo of the complete setup can be seen in Figure 3.3.

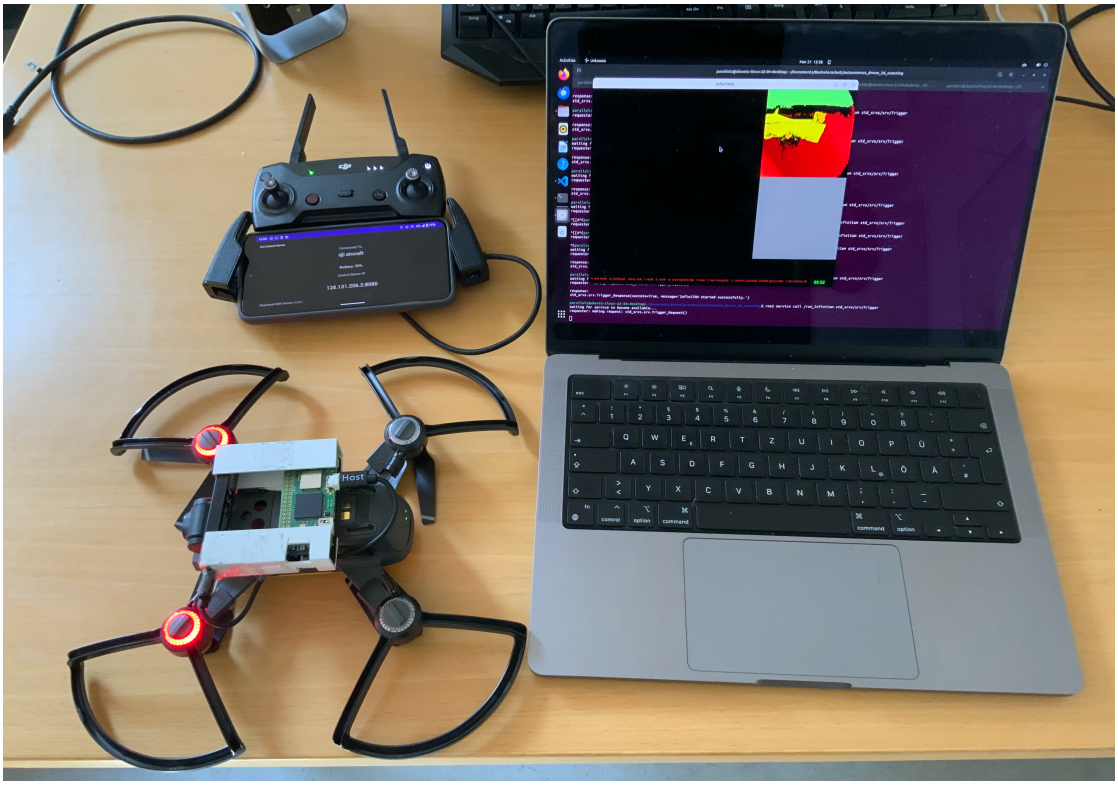


Figure 3.3: Final hardware setup.

3.2.2 Software Setup

Our software architecture is built on ROS 2 and consists of nodes for sensing, mapping, viewpoint planning, and drone navigation. These components are distributed across the Raspberry Pi and the server. They communicate primarily through ROS topics and services. Figure 3.4 shows a simplified ROS graph illustrating the interactions between the main nodes.

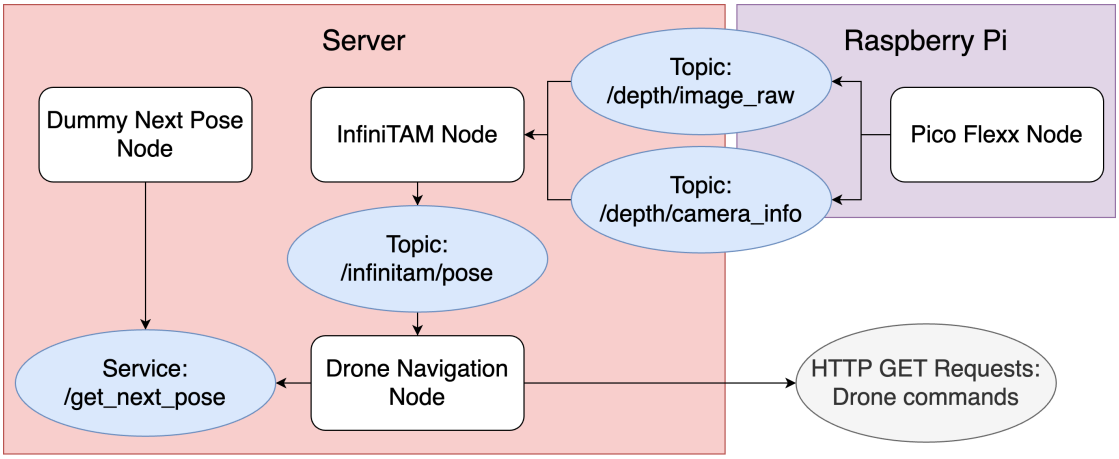


Figure 3.4: Simplified ROS graph showing communication between the main nodes of the implemented system. Only inter-node topics and services are shown.

Pico Flexx Node

This node is responsible for acquiring depth images from the Pico Flexx time-of-flight sensor and publishing them as ROS 2 topics. It interfaces with the Royale SDK and currently publishes the following:

- `/camera/depth/image_raw`: Raw depth image.
- `/camera/depth/camera_info`: Intrinsic parameters of the depth camera, including focal length and distortion coefficients.

InfiniTAM

For our SLAM implementation, we selected InfiniTAM — a real-time 3D reconstruction framework based on the KinectFusion-style pipeline. It uses volumetric fusion with a truncated signed distance function (TSDF) and Iterative Closest Point (ICP) tracking to build dense 3D maps from depth input [KPR+15, KPM16, PKG+17]. Although originally designed for RGB-D data, InfiniTAM can work with depth-only input, which aligns well

with the constraints of our setup and was the main reason we chose it over more modern systems like RTAB-Map [LM19], which we could not get to run without RGB data. We used the basic engine provided by InfiniTAM, which does not support loop closure, as we found it to be a bit more stable for localization.

InfiniTAM-ROS2 Node

This node wraps the InfiniTAM framework to enable depth-only 3D reconstruction within a ROS 2 environment. It uses a custom `ImageSourceEngine` that serves as an entry point for feeding ROS image and camera info messages directly into InfiniTAM's processing pipeline.

The node offers:

- `/run_infinitam` service: Initializes InfiniTAM and launches its user interface.
- `/infinitam/pose` publisher: Publishes the current estimated pose.
- A user interface for manual control and STL mesh export.

Dummy Next-View Node

This placeholder module stands in for the Next-Best-View and Path-Planning components of the planned system. It returns the next viewpoint from a hardcoded list, simulating the behavior of a full exploration stack. It provides the `/get_next_pose` service, which returns a custom message containing:

- `success`: Whether a new pose is available.
- `message`: Debug text.
- `pose`: Target position and orientation.

Drone Navigation Node

This node handles movement command generation. It subscribes to `/infinitam/pose` for the current pose and queries `/get_next_pose` for the target. Its services include:

- `takeoff`, `land`: Basic drone actions via the DJI Control Server.
- `navigate_to_next_pose`: Requests and navigates to the next scanning position, following the procedure described in Algorithms 3.1, 3.2, and 3.3.
- `enable_velocity_control`: Prepares the drone for velocity commands. This service must be called before invoking the navigation service and disables control through the Spark's remote controller.

- `stop_drone`: Stops velocity mode and restores manual control.

The coordinate systems used by InfiniTAM and DJI differ. InfiniTAM uses a left-handed system (x: left, y: up, z: backward), while DJI uses a right-handed one (x: forward, y: right, z: down). We currently manually convert between these systems in code, though a better approach for the future would be to use ROS 2's `tf2` library for managing coordinate transformations dynamically.

DJI Control Server

The DJI Control Server [Kap] app, developed by Dhruv Kapur, exposes DJI's Mobile SDK functionality via RESTful HTTP endpoints. Running on the smartphone, it serves as a bridge between ROS-based control logic and the DJI Spark drone, allowing us to issue commands such as takeoff, landing, and velocity control through simple HTTP GET requests.

3.3 Implementation Limitations

Each major component of the system introduced its own set of limitations, which collectively constrained the performance, stability, and autonomy of the scanning system.

3.3.1 Drone

The DJI Spark was not built to carry external payloads. Even small shifts in mounted hardware affected flight stability, and strict payload limits restricted our choice of sensors and data transmission methods. Additionally, control was only possible via the mobile SDK, which added unnecessary complexity to the system's communication flow.

Depth Sensor and SLAM

The depth-only SLAM approach used in our system struggled in environments with few geometric features. Featureless walls and minimal variation in the scene reduced localization stability and degraded overall scan quality.

The Pico Flexx depth sensor had a limited effective range and narrow field of view. Its connection to the Raspberry Pi was unstable during flight, likely due to vibrations or mechanical stress, making airborne scanning unreliable. The sensor was connected via USB2, although the manufacturer recommends USB3. While we observed no critical issues, we limited ourselves to transmitting at 5 fps. Ultimately, the primary bottleneck was the processing server.

3.3.2 Processing Server

Our server lacked GPU acceleration for InfiniTAM, resulting in slow processing times. This significantly impacted the system's ability to maintain stable localization, especially during motion or rotation.

Algorithm 3.1: Navigate to Next Pose

Input: Current drone pose `current_pose`, Target pose `target_pose`
Output: Drone moves to next target position

if *!next_pose* **then**
  return;
else
  rotateTowardsPose(*current_pose*, *target_pose*);
  moveTowardsPose(*current_pose*, *target_pose*);
end

Algorithm 3.2: Rotate Towards Pose

Input: Current drone pose `current_pose`, Target pose `target_pose`
Output: Drone is oriented to face the target
Global Constants: yaw_threshold, rotation_speed

`/* Yaw calculations include conversion from InfiniTAM's to the drone's coordinate system */`

$dx \leftarrow target\_pose.position.x - current\_pose.position.x$;
$dz \leftarrow target\_pose.position.z - current\_pose.position.z$;
$target\_yaw \leftarrow \texttt{atan2}(-dx, -dz)$;
$q_x, q_y, q_z, q_w \leftarrow current\_pose.orientation$;
$current\_yaw \leftarrow \texttt{asin}(2 \cdot (q_w \cdot q_y - q_z \cdot q_x))$;
$yaw\_error \leftarrow target\_yaw - current\_yaw$;
while $|\ yaw\_error\ | > yaw\_threshold$ **do**
  $yaw\_rate \leftarrow (yaw\_error > 0)\ ?\ rotation\_speed\ :\ -rotation\_speed$;
  $sendRotationCommand(yaw\_rate)$;
  $sleep(500ms)$;
  `/* current_pose is updated by a different thread */`
  $q_x, q_y, q_z, q_w \leftarrow current\_pose.orientation$;
  $current\_yaw \leftarrow \texttt{asin}(2 \cdot (q_w \cdot q_y - q_z \cdot q_x))$;
  $yaw\_error \leftarrow target\_yaw - current\_yaw$;
end

Algorithm 3.3: Move Towards Target Pose

Input: Current drone pose `current_pose`, Target pose `target_pose`
Output: Drone moves to target position
Global Constants: distance_threshold, translation_speed

while $distance(current\_pose,\ target\_pose) > distance\_threshold$ **do**
  $direction \leftarrow normalize(target\_pose.position - current\_pose.position)$;
  $velocity \leftarrow direction \cdot translation\_speed$;
  `/* Convert from InfiniTAM (LH) to DJI (RH) coordinate system */`
  $velocity_{dji} \leftarrow (-velocity.z,\ -velocity.x,\ -velocity.y)$;
  sendMovementCommand($velocity_{dji}$);
  sleep(500ms);
end

CHAPTER 4

Evaluation

This chapter evaluates the physical setup, drone stability, and scanning performance of the proposed system. Alongside performance metrics, it highlights key challenges encountered during development and testing. The goal is to identify practical limitations and assess the system's feasibility for autonomous 3D scanning.

4.1 Mounting the Raspberry Pi

As part of our approach, we needed to attach a Raspberry Pi—or a similarly compact, Wi-Fi and USB-capable board—to the drone. While seemingly straightforward, this presented several unexpected challenges, which we detail in this section.

4.1.1 Payload Capacity and Flight Stability

The DJI Spark does not officially support external payloads; no specifications regarding its carrying capacity are available. Through experimentation, we determined the following limits:

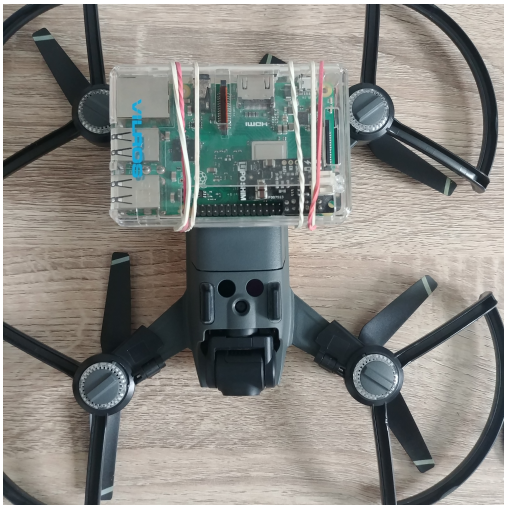
- **Maximum payload capacity:** ~160 g (without propeller guards)
- **Maximum payload with guards:** ~120 g

These payloads allowed the drone to take off reliably; however, any additional weight noticeably reduced its ability to compensate for turbulence.

4.1.2 Initial Attempts with Raspberry Pi 2 Model B+

The initial plan for the hardware included the Raspberry Pi 2 Model B+. While the combination of the Raspberry Pi, sensor, cable, and battery narrowly stayed within the

(a) Raspberry Pi centered. Blocked sensors



(b) Raspberry Pi to the back. Threw the drone out of balance

(c) Wood block with dimensions of Raspberry Pi mounted on its long side. Blocked sensors

(d) Raspberry Pi Zero mounted along the length of the drone. Blocked sensors

Figure 4.1: Overview of failed mounting positions

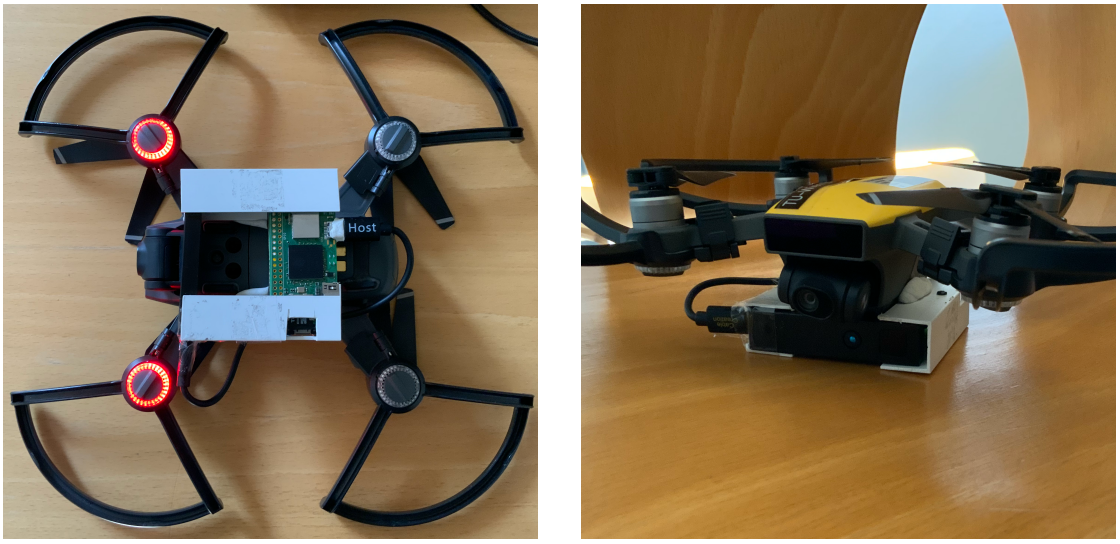


Figure 4.2: Final mounting setup for Raspberry Pi Zero 2 W and Pico Flexx

120 g weight limit, the form factor and weight distribution posed significant challenges. Various mounting strategies were tested, some documented in 4.1, but none proved successful.

However, this initial phase helped us identify several critical conditions that any mounting strategy would need to satisfy:

- **Sensor clearance:** The downward-facing sensors must not be obstructed. If blocked, the drone fails to stabilize at the expected height of approximately 1.5 m after takeoff and continues to ascend uncontrollably—an unacceptable behavior in an autonomous navigation context.
- **Center of mass:** To maintain flight stability, the payload weight must be centered as closely as possible around the drone's center of mass.
- **Mounting robustness:** The mounting method must minimize vibrations, preserve sensor visibility, avoid blocking ventilation holes, and be non-permanent, since the only feasible mounting location is beneath the battery, which must remain removable for battery swaps.

4.1.3 Switching to Raspberry Pi Zero 2 W

With these restrictions in mind, we chose to transition to a Raspberry Pi Zero 2 W. This model significantly reduced the setup's weight to approximately 75 g, detailed in 4.1. Its smaller and lighter design allowed us to find a mounting position that kept the drone as balanced as possible while avoiding the obstruction of any sensors.

For mounting, we initially used cable ties, which—while seemingly secure—applied too much physical stress when fastened tightly. This led to connectivity issues between the sensor and Raspberry Pi, as well as occasional power loss. Ultimately, we settled on a more reliable solution using a combination of adhesive tape and malleable putty, which provided a stable, sensor-friendly, and non-damaging mount. Figure 4.2 shows this final setup.

Component	**Weight (g)**
Cable	5
Pico Flexx	7
Mounting Rails	16
Battery	29
Raspberry Pi Zero + screws	14
Adhesives (approx.)	4
Total	**75**

Table 4.1: Weight breakdown of components mounted on the drone.

4.2 Drone stability

With the hardware securely mounted, we next evaluated how the added payload affected the drone's flight stability. To do so, we conducted a series of repeatable tests measuring deviations in landing positions after basic maneuvers.

Since tracking the drone's behavior in mid-air is difficult, we evaluated stability by measuring the difference between the takeoff and landing positions. Each test was conducted with the Raspberry Pi + Sensor Combo and the bare drone as a control.

We performed the following test sequences:

- **Takeoff-Land:** The drone takes off and lands immediately.
- **Takeoff-Hover-Land:** The drone takes off and hovers for 30 seconds before landing.
- **Takeoff-Fly-Land:** The drone takes off and flies forward 1 meter before landing.

For the takeoff-fly-land tests, we used the `/moveForward/1.0` command from the DJI Control Server. While this command does not provide high precision, it is sufficient for evaluating the relative differences between the bare and loaded drones.

4.2.1 Results and Observations

The distances between takeoff and landing positions of all test sequences are summarized in Tables 4.2, 4.3, and 4.4. Across all test conditions, the drone exhibited greater instability when the payload was attached compared to when it flew unloaded.

Table 4.2: Takeoff-Land Sequence Stability Results

Test Condition	Mean Deviation (cm)	Standard Deviation (cm)
No RPi	9.6	3.6
RPi (Set 1)	37.4	16.8
RPi (Set 2)	11.6	5.9

Table 4.3: Takeoff-Hover-Land Sequence Stability Results

Test Condition	Mean Distance (cm)	Standard Deviation (cm)
No RPi	18.0	9.8
RPi	28.8	18.9

Table 4.4: Takeoff-Fly(1m)-Land Sequence Stability Results

Test Condition	Mean Distance (cm)	Standard Deviation (cm)
No RPi (forward movement)	110.6	8.5
No RPi (sideways movement)	32.0	22.8
No RPi (total distance)	117.0	7.8
RPi (forward movement)	129.6	10.5
RPi (sideways movement)	17.2	6.1
RPi (total distance)	130.9	10.3

For the **Takeoff-Land** tests, we performed two sets of experiments with the RPi attached, differing only in its placement by approximately 2mm. As shown in the results, even this minor adjustment significantly influenced stability. In some cases, the loaded drone performed almost as well as the unloaded drone, while in others, it exhibited noticeably worse stability. Despite efforts to maintain consistent placement, slight deviations were unavoidable due to the need to remove the payload for battery swaps.

In the **Takeoff-Hover-Land** test, the impact of instability became more pronounced. The mean and standard deviations were significantly higher with the payload attached, indicating increased difficulty in maintaining a steady hover.

In the **Takeoff-Fly-Land** tests, an interesting pattern emerged. The unloaded drone exhibited a significant leftward drift (recorded as sideways movement in the table). However, when the payload was attached, the sideways drift was reduced to approximately half, with the standard deviation dropping to about one-fourth of the original. While the loaded drone traveled a greater forward distance on average, the payload placement appeared to mitigate the drone's inherent lateral drift.

Audiovisual and Control Observations

Beyond numerical results, qualitative observations provided additional insights into the drone's behavior:

- **Drift Correction:** During hovering, the drone actively corrected drift, adjusting its position toward the original takeoff point. This correction mechanism was more effective without the payload. With the payload attached, the drone tended to overcorrect drift, contributing to the increased deviation observed in hover test results.
- **Flight Stability:** While hovering, the loaded drone was visibly less stable than the unloaded one, exhibiting more erratic movements.
- **Motor Strain & Performance Limits:** The drone was audibly louder when carrying the payload, suggesting increased motor effort. Additionally, the DJI GO 4 app frequently displayed the warning *"Max motor speed reached"*, indicating that the drone was operating near its performance limits.

4.2.2 Conclusion

Our results show that while the loaded drone can maintain overall stability and compensate for some drift, it operates at its performance limits. Small changes in payload positioning introduce significant instability, making consistent mounting challenging. Since the payload must be removed for battery swaps, achieving a perfectly repeatable balance remains difficult.

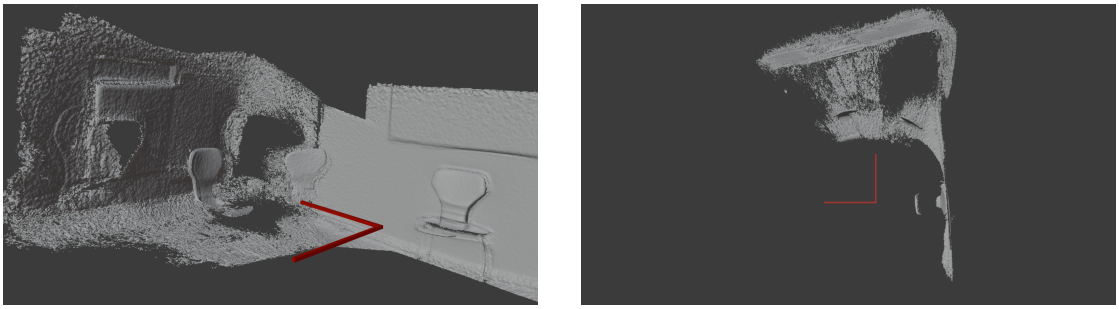
4.3 Scan Process Evaluation

Having assessed the drone's physical stability during flight, we proceeded to evaluate the entire scan process, which revealed several limitations that prevented fully autonomous, room-scale scanning. Ultimately, we were only able to complete a single, partially successful scan — limited to a small corner of a room and involving just two navigation poses.

In the following Section, we evaluate that single scan based on quality and speed, finishing up with the limitations that prevented us from doing more and larger scans. For the evaluation, we compare the drone scan to two alternative scans: one created by manually moving the sensor through the room and another using the "3D Scanner App" [Laa25] on an iPad Pro, which uses its built-in LiDAR sensor.

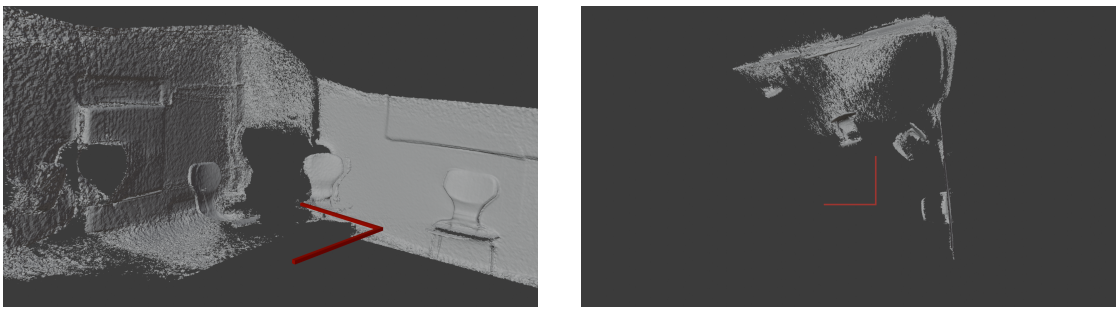
4.3.1 Scan Quality

Figures 4.3, 4.4 and 4.5 show the results of the different scans. The InfiniTAM-based scans—both from the drone and the handheld configuration—exhibited several recurring issues:



(a) Front view (b) Top view

Figure 4.3: Scan results from the autonomous drone scan. *Red line shows approximate trajectory of drone.*



(a) Front view (b) Top view

Figure 4.4: Scan results from the handheld scan. *Red line shows approximate trajectory of sensor.*

(a) Front view (b) Top view

Figure 4.5: Scan results from iPad LiDAR scan.

- **Geometric inaccuracies:** The corner of the room failed to retain a 90-degree angle, suggesting accumulated localization drift. The drone-based scan also seemed to rotate the left wall upwards.
- **Distance-related noise:** Surfaces scanned from closer distances (e.g., the right wall at ∼0.5 m) were noticeably cleaner and more detailed than those farther away (e.g., the left wall at ∼1 m), even though these distances are still well within the sensor's supposed operating range (up to 4 m).
- **Limited reconstruction of horizontal surfaces:** The floor and other surfaces roughly parallel to the drone's navigation plane (e.g., the seats of chairs) were poorly reconstructed. Even after lowering the sensor to ∼0.5 m, the floor remained underrepresented, indicating consistent difficulty capturing horizontal geometry.

4.3.2 Scan Speed

To reduce in-flight instability and improve the chances of successful localization with InfiniTAM, we chose the lowest possible speeds that still produced smooth, continuous drone movement: 0.15 meters per second for linear motion and 10 degrees per second for rotation.

When comparing scan durations, we observe that all methods required a similar amount of time, as summarized in Table 4.5. The handheld scan took slightly longer to ensure that localization remained stable throughout. The iPad scan covered a significantly larger area in a similar duration, though with less detail. For the drone scan, the time spent on takeoff, stabilization, and landing made up a relatively large portion of the total duration compared to the actual scanning time.

Table 4.5: Comparison of scan durations across different methods.

Scanning Method	**Scan Duration**	**Total Duration**
Autonomous Drone Scan	∼30 sec	∼1 min
Handheld Scan	∼1 min	∼1 min
iPad LiDAR Scan	∼50 sec	∼50 sec

Note: Total Duration includes time needed for takeoff and landing.

4.3.3 Challenges and Limitations

This section highlights our main challenges and limitations during this testing step.

Instability of the Depth Sensor Connection

During the flight, maintaining a stable connection between the Pico Flexx depth sensor and the Raspberry Pi proved to be a critical issue. Initially, the sensor disconnected immediately upon takeoff. Switching from cable ties to adhesive mounting improved

this slightly, allowing for 5–30 seconds of usable scanning time. However, the connection still regularly failed mid-flight, and additional attempts to stabilize the USB cable had no measurable effect. Our best guess is that there remains some mechanical stress on the system during flight, possibly caused by vibrations from the drone or air pressure generated by the rotors. Ultimately, this hardware issue remained unresolved, significantly limiting the duration and reliability of data capture.

Unreliable Drone Command Transmission

The DJI Control Server app's method of issuing drone commands via HTTP requests was ultimately unsuitable for real-time navigation. Latencies varied wildly — from a few milliseconds to over 10 seconds — and commands were only reliably processed while the phone's screen was active. During drone operations, these delays led to dangerously unpredictable behavior, including multiple near-collisions. Emergency interventions, such as physically grabbing the drone mid-flight, became necessary — clearly violating both safety and autonomy requirements.

Limited Range of the Depth Sensor

The Pico Flexx sensor, combined with InfiniTAM's internal range filtering (discarding data beyond $\sim$2 m), severely restricted the effective scanning area. Attempts to scan larger portions of the room were unsuccessful, as much of the observed scene exceeded this range and was therefore ignored. As a result, we could only produce a partial reconstruction of a single corner of the room.

Limited Compute Capabilities of the Server

Even during handheld scanning, we encountered significant localization challenges, particularly during rotational movements. In addition to the previously mentioned range limitations, one likely contributing factor was the limited processing power of our server setup. Due to the absence of GPU acceleration, each frame required 300 to 500 ms to process, introducing considerable latency and increasing the likelihood of tracking failure.

Minimal Geometric Features in the Environment

The testing environment offered limited geometric features, further hindering reliable tracking. Large, bare walls provided little structure for the tracker to lock onto. To mitigate this, we strategically placed chairs in the scene to introduce variation and improve localization stability.

4.4 System Comparison

Our proposed system can be considered a hybrid approach, combining strengths from both the ICON drone by Zhang et al. [ZYZ24] and the system proposed by Alexovič et

al [ALB23]. Similar to the ICON drone, our approach supports fully autonomous 3D navigation with online path planning, a capability absent in other discussed systems that either rely on predefined paths, existing maps, or are limited to 2D navigation.

Similar to Alexovič et al., we offload computational tasks by streaming captured data to an external server. This design decision significantly reduces onboard computational requirements, enabling our drone to be lighter and smaller compared to the ICON drone. While precise weight or size details are not provided by Zhang et al., their use of an Intel NUC computer, GoPro 11 mini, and an Intel RealSense D435i camera suggests a considerably higher payload. In contrast, our system's payload is approximately 75 g, allowing for the use of smaller, more agile, and potentially safer drones suitable for indoor environments.

However, streaming to an external server introduces additional latency, which we have not yet evaluated, and requires a stable network connection between the drone and the server, potentially limiting deployment scenarios.

CHAPTER 5

Conclusion

With this thesis, we set out to explore the feasibility of using the DJI Spark—a consumer drone—for fully automated indoor 3D scanning. Although we were able to complete one successful autonomous scan, the overall system proved too limited to be considered feasible for further development. The combination of the Spark's payload constraints and constricting control interface, the limited range and field of view of the Pico Flexx sensor, and the challenges of depth-only SLAM led to frequent issues with flight stability, unreliable sensor connections, poor localization, and slow processing times.

Despite these limitations, we proposed a system architecture and developed a software stack that lays a solid theoretical foundation for future development. The modular design, built on ROS 2 and InfiniTAM, integrates sensing, SLAM, planning, and control components, and can serve as a blueprint for more capable hardware setups.

5.1 Future Work

To overcome the limitations of the current setup, employing a different drone is essential. Other projects—such as [ALB23]-have found success using UAVs compatible with PX4, which would greatly simplify direct control from the server and eliminate the need for indirect smartphone-based relays.

We also recommend switching to a full RGB-D sensor to improve SLAM performance in low-texture and geometrically sparse environments. While InfiniTAM served well for prototyping, a transition to a more modern and widely adopted SLAM framework like RTAB-Map should be considered, especially given its strong ROS integration and larger community.

On the processing side, the server should support GPU-accelerated SLAM to handle real-time reconstruction and localization tasks more efficiently. The current CPU-bound setup introduced significant latency and instability during flight.

Finally, several software components remain to be integrated to achieve full autonomy:

- A proper Next-Best-View planner.
- A path planning module capable of generating paths towards next viewpoints while navigating around obstacles.

Integrating these components, in combination with more capable hardware, would bring the system significantly closer to its original goal of fully autonomous indoor 3D scanning.

Overview of Generative AI Tools Used

Various AI-based tools were employed throughout the development of this thesis. ChatGPT-4o was used during the system development phase, primarily for assistance with ROS 2-related programming and integration tasks. Additionally, ChatGPT-4o and Grammarly were used to refine formulations, improve linguistic clarity, and support proofreading. The Consensus AI platform was utilized to support the literature review process by assisting in the exploration of related research fields.

List of Figures

3.1 Planned system architecture for autonomous indoor 3D scanning. 10
3.2 Overview of the implemented system. Sensor data is published via ROS on the Raspberry Pi and transmitted over the smartphone's hotspot network to the server. The server processes the data and sends flight commands to the smartphone via HTTP GET requests, which relays them to the drones controller via USB. 10
3.3 Final hardware setup. 11
3.4 Simplified ROS graph showing communication between the main nodes of the implemented system. Only inter-node topics and services are shown. . . . 12

4.1 Overview of failed mounting positions 18
4.2 Final mounting setup for Raspberry Pi Zero 2 W and Pico Flexx 19
4.3 Scan results from the autonomous drone scan. *Red line shows approximate trajectory of drone.* 23
4.4 Scan results from the handheld scan. *Red line shows approximate trajectory of sensor.* 23
4.5 Scan results from iPad LiDAR scan. 23

List of Tables

4.1 Weight breakdown of components mounted on the drone. 20
4.2 Takeoff-Land Sequence Stability Results 21
4.3 Takeoff-Hover-Land Sequence Stability Results 21
4.4 Takeoff-Fly(1m)-Land Sequence Stability Results 21
4.5 Comparison of scan durations across different methods. 24

List of Algorithms

3.1 Navigate to Next Pose 15
3.2 Rotate Towards Pose 15
3.3 Move Towards Target Pose 16

Bibliography

[AFDT22] Iman Abaspur Kazerouni, Luke Fitzgerald, Gerard Dooly, and Daniel Toal. A survey of state-of-the-art on visual slam. *Expert Systems with Applications*, 205:117734, 2022.

[ALB23] Stanislav Alexovič, Milan Lacko, and Ján Bačík. 3d mapping with a drone equipped with a depth camera in indoor environment. *Acta Electrotechnica et Informatica*, 23:18–24, 06 2023.

[AMS+19] Ben Ayawli, Xue Mei, Moquan Shen, Albert Appiah, and Rev. Engr. Frimpong Kyeremeh. Optimized rrt-a* path planning method for mobile robots in partially known environment. *Information Technology And Control*, 48:179–194, 06 2019.

[AR24] Ahmad A. Bany Abdelnabi and G. Rabadi. Human detection from unmanned aerial vehicles' images for search and rescue missions: A state-of-the-art review. *IEEE Access*, 12:152009–152035, 2024.

[ARN+24] Iyer Ashwin, Manojkumar Rajagopal, M. Naren, V. Santosh Narayan, and Bala Murugan. Autonomous systems: Indoor drone navigation. In *American Institute of Physics Conference Series*, volume 3216 of *American Institute of Physics Conference Series*, page 020002. AIP, July 2024.

[BDRM24] Jorge Bes, Juan Dendarieta, Luis Riazuelo, and Luis Montano. Dwa-3d: A reactive planner for robust and efficient autonomous uav navigation. *arXiv pre-print arXiv:2409.05421*, 2024.

[BPI+21] Ana Batinovic, Tamara Petrovic, Antun Ivanovic, Frano Petric, and Stjepan Bogdan. A multi-resolution frontier-based planner for autonomous 3d exploration. *IEEE Robotics and Automation Letters*, 6(3):4528–4535, 2021.

[CER+21] Carlos Campos, Richard Elvira, Juan J. Gómez Rodríguez, José M. M. Montiel, and Juan D. Tardós. Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021.

[DTMD08] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Practical search techniques in path planning for autonomous driving. *AAAI Workshop - Technical Report*, 01 2008.

[EKC16] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP, 07 2016.

[ENCA17] Milan Erdelj, E. Natalizio, K. Chowdhury, and I. Akyildiz. Help from the sky: Leveraging uavs for disaster management. *IEEE Pervasive Computing*, 16:24–32, 2017.

[ESC14] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 834–849, Cham, 2014. Springer International Publishing.

[HH11] Sebastian Haner and Anders Heyden. Optimal view path planning for visual slam. In Anders Heyden and Fredrik Kahl, editors, *Image Analysis*, pages 370–380, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[Kap] Dhruv Kapur. Dji control server: Embedded rest server for controlling dji drones. `https://github.com/dkapur17/DJIControlServer`, Accessed: 2025-03-16.

[KHDK22] Geesara Kulathunga, Hany Hamed, Dmitry Devitt, and Alexandr Klimchik. Optimization-based trajectory tracking approach for multi-rotor aerial vehicles in unknown environments. *arXiv pre-print arXiv:2202.06056*, 2022.

[KPFV21] Sander Krul, Christos Pantos, Mihai Frangulea, and João Valente. Visual slam for indoor livestock and farming using a small drone with a monocular camera: A feasibility study. *Drones*, 5(2), 2021.

[KPM16] Olaf Kähler, Victor Adrian Prisacariu, and David W. Murray. Real-time large-scale dense 3d reconstruction with loop closure. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII*, pages 500–516, 2016.

[KPR+15] O. Kähler, V. A. Prisacariu, C. Y. Ren, X. Sun, P. H. S Torr, and D. W. Murray. Very High Frame Rate Volumetric Integration of Depth Images on Mobile Device. *IEEE Transactions on Visualization and Computer Graphics (Proceedings International Symposium on Mixed and Augmented Reality 2015*, 22(11), 2015.

[Laa25] Laan Labs. 3D Scanner App (Version 1.1.6), 2025. Mobile application, available on the Apple App Store, `https://apps.apple.com/at/app/3d-scanner-app/id1419913995?l=en-GB`, Accessed: 2025-03-20.

[LDLMT22] Liang Lu, Alessio De Luca, Luca Muratore, and Nikos G Tsagarakis. An optimal frontier enhanced "next best view" planner for autonomous exploration. In *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*, pages 397–404, 2022.

[LM19] Mathieu Labbé and François Michaud. Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*, 36(2):416–446, 2019.

[MFG+22] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022.

[MVGT+20] Miguel Mendoza, J. Irving Vasquez-Gomez, Hind Taud, L. Enrique Sucar, and Carolina Reta. Supervised learning of the next-best-view for 3d object reconstruction. *Pattern Recognition Letters*, 133:224–231, 2020.

[NIH+11] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, 2011.

[PKG+17] V A Prisacariu, O Kähler, S Golodetz, M Sapienza, T Cavallari, P H S Torr, and D W Murray. InfiniTAM v3: A Framework for Large-Scale 3D Reconstruction with Loop Closure. *arXiv pre-print arXiv:1708.00783v1*, 2017.

[QCG+09] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, 2009.

[QYS21] Jie Qi, Hui Yang, and Haixin Sun. Mod-rrt*: A sampling-based algorithm for robot path planning in dynamic environment. *IEEE Transactions on Industrial Electronics*, 68(8):7244–7251, 2021.

[STD+19] Magnus Selin, Mattias Tiger, Daniel Duberg, Fredrik Heintz, and Patric Jensfelt. Efficient autonomous exploration planning of large-scale 3-d environments. *IEEE Robotics and Automation Letters*, 4(2):1699–1706, 2019.

[Ste94] A. Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 3310–3317 vol.4, 1994.

[WCL+20] Jiankun Wang, Wenzheng Chi, Chenming Li, Chaoqun Wang, and Max Q.-H. Meng. Neural rrt*: Learning-based optimal path planning. *IEEE Transactions on Automation Science and Engineering*, 17(4):1748–1758, 2020.

[WLSM+15] Thomas Whelan, Stefan Leutenegger, Renato F Salas-Moreno, Ben Glocker, and Andrew J Davison. Elasticfusion: Dense slam without a pose graph. In *Robotics: science and systems*, volume 11, page 3. Rome, 2015.

[WXC+24] Tao Wang, Weibin Xi, Yong Cheng, Hao Han, and Yang Yang. Rl-nbv: A deep reinforcement learning based next-best-view method for unknown object reconstruction. *Pattern Recognition Letters*, 184:1–6, 2024.

[XMW+21] Ronglei Xie, Zhijun Meng, Lifeng Wang, Haochen Li, Kaipeng Wang, and Zhe Wu. Unmanned aerial vehicle path planning algorithm based on deep reinforcement learning in large-scale and dynamic environments. *IEEE Access*, 9:24884–24900, 2021.

[Yam97] B. Yamauchi. A frontier-based approach for autonomous exploration. In *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*, pages 146–151, 1997.

[ZXZ+22] Yao Zhao, Zhi Xiong, Shuailin Zhou, Jingqi Wang, Ling Zhang, and Pascual Campoy. Perception-aware planning for active slam in dynamic environments. *Remote Sensing*, 14(11), 2022.

[ZYZ24] Hao Xuan Zhang, Yilin Yang, and Zhengbo Zou. Icon drone: Autonomous indoor exploration using unmanned aerial vehicle for semantic 3d reconstruction. In *Proceedings of the 11th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, BuildSys '24, page 66–76, New York, NY, USA, 2024. Association for Computing Machinery.