TU WIEN Informatics

Building a drone platform for autonomous scanning

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Jonathan Proelss
Matrikelnummer 11912403

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, 6. August 2025

Jonathan Proelss	Michael Wimmer

TU WIEN Informatics

Building a drone platform for autonomous 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

Jonathan Proelss
Registration Number 11912403

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, August 6, 2025

Jonathan Proelss	Michael Wimmer

Erklärung zur Verfassung der Arbeit

Jonathan Proelss

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, 6. August 2025

Jonathan Proelss

Danksagung

Ich möchte mich herzlich bei meiner Familie und meinen Freunden bedanken, die mir immer beigestanden sind und mich bei der Anfertigung dieser Bachelorarbeit unterstützt und motiviert haben

Mein besonderer Dank gilt meinem Betreuer Mag.rer.soc.oec. PhD Stefan Ohrhallinger, der mich mit kontinuierlichem und konstruktivem Feedback durch den gesamten Prozess begleitet hat.

Acknowledgements

I would like to express my y heartfelt gratitud to my family and friends, who have always stood by me and supported and motivated me in the preparation of this bachelor thesis.

Special thanks go to my supervisor, Mag.rer.soc.oec. PhD Stefan Ohrhallinger, who guided me through the entire process with continuous and constructive feedback.

Kurzfassung

In den letzten Jahren sind unbemannte Luftfahrzeuge (UAVs) unter anderem aufgrund sinkender Herstellungskosten immer populärer geworden. Sie finden sowohl im Unterhaltungsbereich als auch in Einsatzszenarien Anwendung. Die meisten für den Außeneinsatz konzipierten Drohnen nutzen typischerweise eine GPS-gestützte Positionsbestimmung, die jedoch in Innenräumen oder Umgebungen mit Hindernissen unzuverlässig ist. In dieser Arbeit wird die Herausforderung der autonomen Indoor-Navigation und Kartierung durch das Design, der Bau und die Evaluierung einer kompakten, modularen Drohnenplattform mit 3D-Scanning-Hardware und integriertem Computer zur Echtzeit-Kartenerstellung und autonomen Navigation behandelt.

Die entwickelte Plattform integriert Visual-Inertial-Odometrie und Tiefenbilder in eine ROS-Noetic-Softwarearchitektur. Die Auswahl der Komponenten, die Sensorkalibrierung, das Energiemanagement und die Softwareintegration wurden systematisch behandelt, um der Plattform kurzzeitige autonome Einsätze zu ermöglichen. Aufgrund einiger Herausforderungen bezüglich der Stabilität der Lokalisierung und der Sensorintegration konnten jedoch keine vollautonomen Flüge abgeschlossen werden.

Abstract

Over the past decade, unmanned aerial vehicles (UAVs) have grown in popularity, primarily due to reduced manufacturing costs. They have a wide range of applications, from entertainment to critical emergency services. However, most drones designed for outdoor navigation rely on GPS-based positioning, which can be unreliable in indoor or obstructed environments. This thesis addresses this challenge by proposing the design, implementation, and evaluation of a compact, modular drone platform equipped with 3D scanning hardware and an onboard computer for real-time mapping and autonomous navigation.

The developed platform integrates visual-inertial odometry and depth imaging within an ROS-Noetic software architecture. The selection of components, sensor calibration, power management, and software integration were all systematically addressed in order to enable short-duration autonomous operations. However, due to challenges regarding localisation stability and sensor integration, fully autonomous missions could not be completed.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Over the past decade, unmanned aerial vehicles, better known as drones, have seen a rapid increase in both affordability and capability. Advances in manufacturing and electronics have reduced the costs, while a wave of innovative applications has emerged. Now, drones are used for entertainment purposes where hundreds of drones perform choreographed light shows [EHa], for life-saving services, such as rapid delivery of medical supplies to remote regions, [CBH+24], and even for firefighting where UAVs are designed to detect and extinguish fires in hazardous environments[JNS+24].

Many of these UAVs are quite large and designed to fly outside autonomously, supported by satellite positioning systems and waypoint navigation. While this approach excels in open skies and over long distances, it breaks down in GPS-denied environments like indoor spaces or urban environments in which walls and obstacles obstruct satellite signals and demand alternative localization and control strategies. So platforms designed to operate indoors or in confined urban settings still often rely on human pilots, who must steer the drone manually through tight passages and avoid obstacles.

Fully autonomously navigating and mapping UAVs could help in many different situations. They could, for example, swiftly generate accurate floor plans of an entire house to streamline renovation projects, thereby eliminating the need for manual measurements. In emergency scenarios, such as a collapsed building or an area with unreliable radio links, these UAVs could independently explore hazardous or RF-blocked environments and provide up-to-date 3D maps to rescue teams.

1.2 Objective

This thesis aims to take a small step in this direction by designing, implementing, and evaluating a compact, modular drone platform that is capable of carrying 3D-scanning equipment and a computer for mapping and autonomous navigation. Acknowledging that realizing every aspect exceeds the scope of this work, we here focus on designing and building the platform, while leaving room for future development and improvements.

The following part of the thesis is structured as follows: Chapter 2 reviews the state of the art in indoor positioning, calibration techniques, middleware frameworks, flight controllers, and autonomous drone scanning. Chapter 3 details our system's design and implementation, from hardware selection and mechanical integration to software architecture and ROS node organization. Chapter 4 presents an evaluation of the platform, examining flight performance, scanning quality, and some practical challenges we encountered. Finally, Chapter 5 summarizes the findings and outlines directions for future work.

CHAPTER 2

Related Work

This chapter gives an overview of the relevant related work used in this thesis.

2.1 Non-GPS Positioning Systems

Robots out in the open sky or roaming a field can lean on satellite-based systems like GPS or Galileo to collect highly accurate position data. This can be problematic in indoor environments because walls can decrease the reliability of the signal. To address this challenge, a variety of alternative localization methods have emerged, including visual-inertial odometry, ultra-wide-band ranging, and LiDAR-based SLAM, which fuse data from onboard sensors to maintain precise position estimates in GPS-denied environments.

2.1.1 ToF based Systems

Time-of-flight (ToF) based positioning systems mirror GPS by measuring signal travel times between fixed anchors and the tracked object. They use multiple stationary anchor stations to triangulate the position. Their coverage is inherently limited by the number and distribution of anchors and the frequencies used. Ultrasonic ToF-solutions often use 40kHz carrier waves and can, in theory, achieve sub-centimeter accuracy over a few meters. Their performance suffers from reflections and ambient noise. Ultra-wide-band applications (UWB) use frequencies from 0.5-10GHz to send short pulses for triangulation. They can provide sub-centimeter accuracy over multiple meters and are more robust than ultrasonic systems.

However, in real-world and Non-Visual-Line-of-Sight (NVLos) conditions, these systems can have high error rates and an accuracy of 1-2m [CSSC+22].

2.1.2 Visual based Systems

Visual-based positioning systems use one or multiple image streams to track the camera's position. They begin by detecting distinctive visual features such as corners, edges, or textured patches in each frame using algorithms like ORB ([RRKB11]) or SIFT ([Low99]), followed by matching those features across successive images (and between multiple cameras, in stereo setups). By tracking the movement of these features over time and using geometric relationships (e.g., triangulation in stereo or motion-based re-projection), the system computes the change of the camera's position and orientation.

Visual–Inertial Odometry (VIO) refers to software systems that fuse high-frequency IMU measurements with images from a monocular or stereo camera to continuously estimate a platform's position, orientation, and velocity. By feeding both visual feature tracks and inertial readings into an Extended Kalman Filter (EKF), VIO-Systems achieves robust, real-time pose estimation with reduced drift compared to purely visual or inertial methods. Extended Kalman Filters were first introduced in 1962 [MG62] based on the Kalman Filter [Kal60], first published by Imre Kalman in 1960.

2.2 Camera IMU Calibration

Reliable VIO requires precise alignment between the camera and IMU sensors' data. Early methods treated spatial and temporal calibration separately, using checkerboard patterns for camera intrinsics and manual synchronization procedures for IMU timing [Zha00]. More recent approaches, most notably the Kalibr toolbox, involve jointly optimizing camera intrinsics, stereo extrinsics, IMU-to-camera rigid transforms, and inter-sensor time offsets. This is achieved by observing a calibration board (for example, one with an AprilTag grid) undergoing various motions [FRS13, RNS+16]. These Unified calibration frameworks have been shown to reduce reprojection errors to below 0.2px, reducing timestamp misalignment to under a few milliseconds and significantly improving filter consistency and reducing drift in VIO systems.

2.3 ROS

Robotic platforms typically follow a modular, decentralized approach that integrates a variety of sensors and specialized software to perform perception, planning, and control tasks. To route each data stream reliably to its intended component, these systems rely on so-called middleware frameworks. The Robot Operating System (ROS) [QCG+09], along with its predecessor ROS 2 [MFG+22], has become a widely adopted solution for managing inter-module communication and orchestration in both research and industry. ROS is built around the following core components:

- **Nodes.** Individual pieces of software that perform computation. Each node registers with the ROS Master and can publish or subscribe to data streams (called topics), call services, or provide services of its own.

- **Topics.** Publish/subscribe buses over which nodes exchange strongly-typed messages. Publishers push data out on a topic; subscribers receive each message as it arrives, decoupling producers from consumers.

- **Services.** Synchronous, RPC-style communication channels. A node advertises a service name and message schema (request + response); clients send a request and block until the server returns a reply, useful for parameter lookups or one-off commands.

2.4 Flight Controller Systems

This section provides an overview of some of the most common flight-control systems used for Unmanned Aerial Vehicles (UAVs).

2.4.1 Ardupilot

ArduPilot is an open-source autopilot software suite designed to control a wide variety of unmanned vehicles, including multicopters, rovers, and boats. Since its first release in 2009 [Com09], it has undergone continuous development driven by a global community of contributors.

It supports a wide variety of sensors and processing algorithms, including inertial measurement units (which combine accelerometers, gyroscopes and magnetometers) for attitude and motion estimation, barometric pressure sensors for altitude hold, GNSS receivers (GPS, GLONASS and BeiDou) for global positioning, and distance sensors such as LiDAR, sonar and ultrasonic rangefinders for obstacle detection, terrain following and advanced obstacle avoidance. It also supports optical-flow or vision-based systems for motion tracking, position hold, and advanced obstacle avoidance. These inputs are fed into modular control loops and navigation stacks, mixed and subsequently used by various unmanned vehicle applications.

2.4.2 Betaflight

Betaflight [B] is an open-source flight-controller firmware specifically engineered for multi-rotor platforms, particularly for first-person view (FPV), racing, and freestyle applications. It is designed to run on 32-bit micro-controllers and emphasizes low-latency control loops.

By limiting its supported sensors to inertial measurement units (IMUs) (i.e., accelerometers and gyroscopes), as well as to other basic sensors, such as barometers and compasses, Betaflight minimizes computational overhead to achieve short control response times. Unlike full-featured autopilot stacks, such as Ardupilot, it omits global navigation algorithms and waypoint trajectory planning, instead relying on human pilot inputs to control the flight path directly.

2.4.3 INav

INav [C] is an open-source flight controller firmware that is a derivative of Cleanflight and Betaflight. It has a stronger focus on autonomous navigation and multi-copter/fixed-wing applications. Like Betaflight, it runs on 32-bit micro-controllers and supports a broader range of sensors, such as GPS and optical flow. Integrating these additional sensors enables advanced autonomous flight capabilities, particularly position hold and waypoint-based navigation.

2.5 Autonomous Drone Scanning Research

In this section, we review previous research employing similar indoor-scanning approaches.

Zhang et al. [ZYZ24] presented the ICON drone, an autonomous indoor UAV system. The drone captures RGB-D imagery and reconstructs accurate 3D geometry, semantically labeling building components and materials, thereby automating BIM creation. It uses visual-inertial odometry (VIO) and frontier-based planning to explore spaces quickly. Furthermore, the drone employs both classical photogrammetry and a vision transformer model for reconstruction, and back-projects Swin-Transformer segmentation onto the point cloud. In tests by the research team, it achieved sub-10 cm reconstruction accuracy and over 80% segmentation accuracy.

In 2018, Dowling et al. [DPH+18] introduced URSA, a tethered indoor UAV system that combines off-board 2D LiDAR SLAM and a fixed base station for mapping and planning. URSA also features an onboard PX4 flight controller, which is fused with IMU and ultrasonic height data, to achieve autonomous navigation and live mapping with an accuracy of ± 2 cm. In real-world tests, URSA reliably avoided static and dynamic obstacles, managed corners and entered narrow doorways, producing 2D maps that were within ± 2 cm of ground-truth tape measurements. While the authors report that performance is solid at low speeds, higher-speed flights reveal overshoot and stability issues. This motivates future work on trajectory simulation with inertia, tighter SLAM–controller integration, and moving towards full 3D mapping.

Most recently, Tukan et al. [TFG+23] presented a real-time autonomous indoor exploration system for toy drones equipped with only a monocular RGB camera. This system leverages ORB-SLAM3 for sparse mapping and localization. They developed a provable submodular outlier removal algorithm to clean the point cloud and a novel 360° 'exit detection' method to guide continuous exploration. The cleaned sparse map was transformed into a LiDAR-like 2D representation using clustering and convex hull construction. This enabled RRT-based path planning with post-hoc path smoothing. Experiments conducted with DJI Tello drones in environments where GPS was unavailable demonstrated reliable map cleaning, effective exit-point selection, and successful navigation through unknown spaces.

CHAPTER 3

Methods

This chapter outlines the construction process of the drone, including the selection of components and why they were chosen. The aim was to create a modular platform that scans autonomously and reconstructs indoor environments. The platform was intended to be easily updated with new software or upgraded hardware in the future.

3.1 Requirements

As the drone will be flying in tight indoor spaces, it needs to be able to tolerate minor bumps and be quick and inexpensive to repair. At the same time, it must be as compact and lightweight as possible, while being able to carry a payload for 4–8 minutes of flight time. It must be capable of fully autonomous indoor flight and navigation using only its onboard sensors, without depending on external positioning systems (e.g., OptiTrack). It should also offer an instant manual override option, enabling a pilot to take control whenever necessary to enhance safety and minimize potential damage. Scanning should happen on the drone itself to reduce computational cost, without live-streaming the data to a server.

3.2 Hardware

As shown in Figure 3.1, the drone architecture comprises two independent subsystems. The flight system, which is powered by its own battery, is responsible for all real-time stabilization and propulsion. It can fly either via the autopilot of the flight controller or manually via remote control. Likewise, the companion computer, which is powered by a separate battery, handles the scanning payload and calculates the next best scanning position. By isolating their power supplies, we ensure that voltage dips, most likely caused by flight maneuvers, do not affect the companion computer. Communication

between the two subsystems is maintained via a single dedicated data link using the MAVLink protocol.

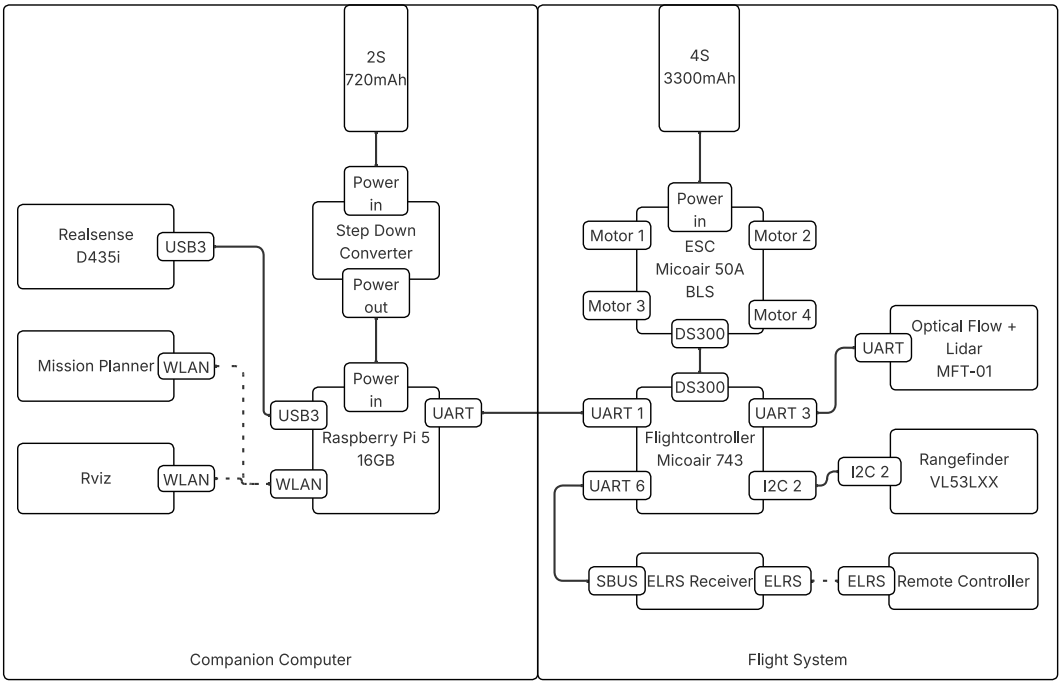


Figure 3.1: Cabling of the drone

3.2.1 Flight System

The first thing we considered was the drone's overall size, which is largely determined by the diameter of the propellers. While large props usually increase payload capacity and stability, they require larger motors and batteries and, due to their size, are more difficult to maneuver in confined spaces. We decided on 3-inch propellers as these allowed us to build a relatively small drone with enough headroom to carry the scanning hardware and achieve a thrust-to-weight ratio of at least 2:1. [HAJ+21] The chosen motors generate a maximum thrust of 845 g each when used with the recommended 3-inch propeller and 4S battery. To prevent overheating and burning out, we calculated with 80% of that, resulting in a maximum take-off weight of:

$$\frac{845\,\mathrm{g} \times 80\% \times 4Motors}{2} = 1352g$$

We chose ArduPilot as our autopilot platform because of its flexibility and broad sensor support, as well as its ability to connect to a companion computer for more advanced in-flight control. As shown in Figure 3.1, we mounted an optical flow sensor on the underside of the drone. This detects apparent movement of the ground below, enabling the

drone to hold its position in environments where GPS is unavailable. The rangefinder is mounted on top of the frame and points upwards to detect the ceiling, thereby preventing accidental crashes and ensuring a safe distance is maintained. Additionally, we have equipped the drone with an ExpressLRS (ELRS) receiver to provide a robust, low-latency radio link, ensuring that the pilot can take over instantly if necessary.

3.2.2 Companion Computer

The onboard computer is a Raspberry Pi 5 with 16 GB of RAM. Weighing just 46 g and with a small form factor, it can be fitted in the middle of the frame without interfering with the propeller. We CNC-milled a custom connector plate from PVC to mount it, which also houses the step-down converter and allows the Pi to be powered directly from a LiPo battery. This was essential, as insufficient power causes the Raspberry Pi 5 to reduce both its CPU performance and USB output. The Raspberry Pi can connect to a Wi-Fi network to monitor the scanning process and the flight controller. For odometry and real-time 3D mapping, we selected the Intel RealSense D435i camera, as it performed well in the aforementioned paper [ZYZ24]. It combines two global-shutter infrared sensors to ensure accurate depth measurements during fast motion, with a rolling-shutter RGB sensor to capture color imagery and an integrated IMU to provide high-frequency accelerometer and gyro data. It uses an infrared projector that beams a semi-random dot pattern onto the environment to increase depth accuracy. The Camera has an optimal operating area of 0.3m - 3.0m [Int].

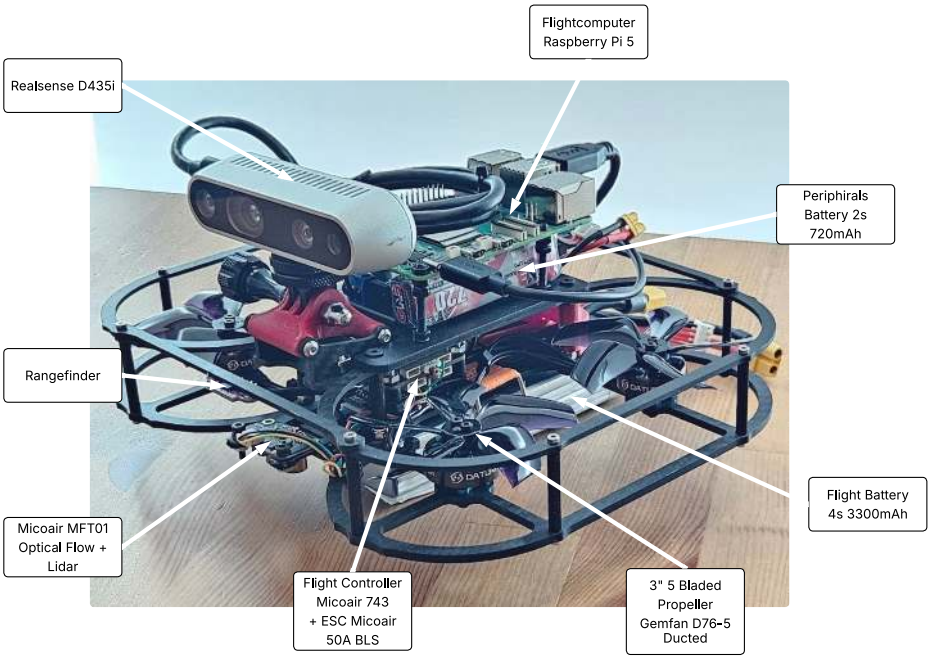


Figure 3.2: Drone fully built

3.2.3 Hardware List

A complete List of all Hardware used to build the drone

Description	Component
Motor	Foxeer Datura 2105.5 3650Kv
Propeller	Gemfan D76-5 Ducted
Frame	Rotorama Bouncer
Drone Battery	GNB 3300mAh 4S 100C
Flight Control Unit (FCU)	MicoAir H743
Electronic Speed Controller (ESC)	MicoAir 50A BLS
ELRS	Radiomaster RP1 2.4GHz
Capacitor	680 µF 50V
Optical Flow + ToF sensor	MicoAir MFT01
ToF sensor	VL53L0X
Remote Controller	Radiomaster Pocket ELRS
Companion Computer	Raspberry Pi5
Computer Battery	GNB 720mAh 2S 100C HV
Step Down Converter	APKLVSR DC-DC 24V/12V to 5V 5A
Stereo Camera	Realsense D435i

Table 3.1: Component List

3.3 Software

We chose ROS Noetic as our main framework for orchestrating all sensor data and control nodes. Due to compatibility issues, it runs on the Raspberry Pi inside a Docker container. Figure 3.3 illustrates the workspace, showing how the nodes are organized and how data flows between them.

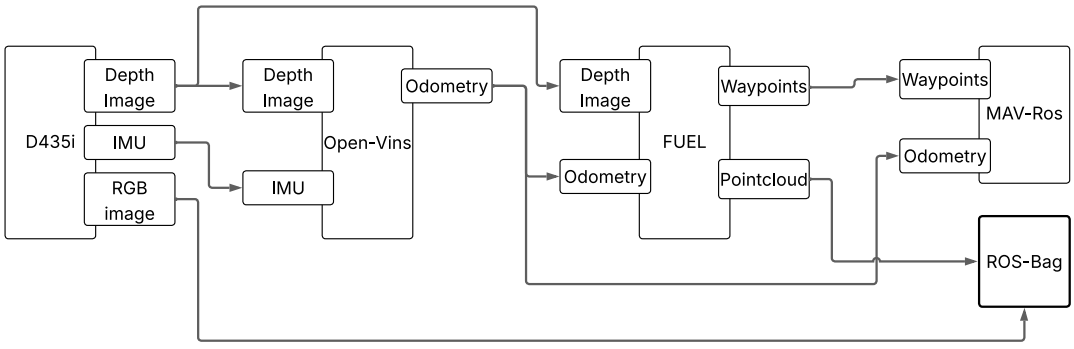


Figure 3.3: ROS Workspace

3.3.1 ROS Workspace

As the drone's main purpose is to 3D scan indoor environments autonomously, we needed a suitable framework to enable communication between the drone platform components. The initial idea was to use a SLAM (Simultaneous Localization and Mapping) system and add a Next Best View algorithm to suggest subsequent scanning locations. However, in practice, this modular approach still required a separate obstacle-free path planner between viewpoints, which added unnecessary complexity. Inspired by the work in [ZYZ24], which uses the FUEL framework to integrate mapping and view planning into a single process [ZZCS21], we therefore adopted FUEL for its 3D scanning capabilities and trajectory planning for the Next Best View. Due to compatibility issues when running the FUEL framework on newer, unsupported ROS and Ubuntu versions, we stuck to older ROS Noetic and Ubuntu 20.04.

Since FUEL relies on accurate external odometry, we integrated the Open VINS system [GEL+20] into our system. When running in a stereo-inertial configuration with the RealSense D435i, Open VINS continuously outputs odometry data to FUEL and the Ardupilot flight controller. To increase accuracy, we calibrated the stereo camera and the IMU using Kalibr [FRS13, FBS12, MFS13]. During testing, we discovered that the light pattern cast by the RealSense infrared projector (IR projector) was disturbing the visual-inertial odometry and causing Open VINS to calculate incorrect motion estimates even when the drone did not move. Disabling the IR projector reduced these errors and stabilized the pose estimates. Without the IR projector, the resulting depth images suffered from increased noise and decreased accuracy. An Interlaced Approach where the projector gets activated every second Frame was not possible with the provided driver of the manufacturer.

To enhance flight safety and proactively prevent collisions, we additionally implemented a 'virtual' rangefinder in the form of a Python node in ROS 3.1. The node continuously analyses the depth image stream from the RealSense camera and extracts the closest distance of the centered image segment. This proximity value is then sent to the flight controller to prevent collisions.

Listing 3.1: Min distance node

```
depth_array = np.frombuffer(nput_topic.data,dtype=np.uint16).reshape
    ((input_topic.height, input_topic.width))
depth = depth_array.astype(np.float32) / 1000.0

crop_size = 100
cx, cy = input_topic.width // 2, input_topic.height // 2
half_crop = crop_size // 2

center_crop = depth[cy - half_crop:cy + half_crop, cx - half_crop:cx
    + half_crop]

valid = center_crop[(center_crop > 0.1) & (center_crop < 10.0)]
if valid.size > 0:
```

```
    minDistance = float(np.min(valid))
    msg = Range()
    msg.radiation_type = Range.INFRARED
    msg.field_of_view = 0.5
    msg.min_range = 0.1
    msg.max_range = 5.0
    msg.range = minDistance
    msg.header.stamp = rospy.Time.now()
    pub.publish(msg)
```

3.3.2 Converter Node

As ArduPilot cannot process Open VINS' odometry messages directly, we developed a dedicated ROS node that translates the data into the format required by ArduPilot. This node also reduces the update rate from approximately 200 Hz to 20 Hz, ensuring that the telemetry link remains within its available bandwidth and avoiding packet loss.

Listing 3.2: Part of the node to send odometry to the flight controller

```
class OdomConverter:
    def __init__(self):
        rospy.init_node("odometry_converter")
        self.pub = rospy.Publisher("/mavros/vision_pose/pose",
                                   PoseStamped, queue_size=1)
        self.last_odom = None
        rospy.Subscriber("/ov_msckf/odomimu", Odometry, self.
            odom_callback)
        self.timer = rospy.Timer(rospy.Duration(1.0/20.0),
                                 self.timer_callback)
        rospy.loginfo("Odometry␣converter␣running␣at␣20␣Hz")
        rospy.spin()

    def odom_callback(self, odom: Odometry):
        # store the most recent odom, don't publish here
        self.last_odom = odom

    def timer_callback(self, event):
        if not self.last_odom:
            return
        ps = PoseStamped()
        ps.header.stamp    = self.last_odom.header.stamp
        ps.header.frame_id = "map"
        ps.pose            = self.last_odom.pose.pose
        self.pub.publish(ps)
        rospy.logdebug(f"Published␣pose␣at␣{ps.header.stamp.to_sec()
            :.3f}")
```

3.3.3 Waypoint publisher

In order to control the drone's trajectory, we needed to feed in the next scan position published by Ardupilot. Ardupilot can operate in two autonomous flight modes.
The first mode requires a list of waypoints. When the mission starts, the drone flies to each waypoint in the respective order. When we operated the drone in this mode, we recognized that the waypoints could not be updated during the mission when FUEL recalculated the trajectory.
As continuous updates are necessary, we tested the second mode, which only accepts one point at a time but ensures continuous updates. In order to control the flight trajectory, we wrote a Python ROS node to convert the FUEL position commands into Ardupilot waypoints.3.3

Listing 3.3: Part of the node that controls the drone by managing the waypoints

```
class TrajBridge:
    def __init__(self):
        # parameters
        in_topic       = rospy.get_param("~in_topic",  "/planning/
            pos_cmd")
        out_topic      = rospy.get_param("~out_topic", "/mavros/
            setpoint_raw/local")
        self.rate_hz   = rospy.get_param("~pub_rate", 15.0)
        self.min_dt    = 1.0 / self.rate_hz
        self.last_pub  = rospy.Time(0)

        # pub/sub
        self.pub = rospy.Publisher(out_topic, PositionTarget,
            queue_size=1)
        rospy.Subscriber(in_topic, PositionCommand, self.cb)

    def cb(self, msg):
        now = rospy.Time.now()
        # throttle to ~15 Hz
        if (now - self.last_pub).to_sec() < self.min_dt:
            return
        self.last_pub = now

        pt = PositionTarget()
        pt.header         = Header(stamp=now, frame_id="map")
        pt.coordinate_frame = PositionTarget.FRAME_LOCAL_NED
        # ignore accelerations
        pt.type_mask = (
            PositionTarget.IGNORE_AFX |
            PositionTarget.IGNORE_AFY |
            PositionTarget.IGNORE_AFZ
        )

        # fill in position
```

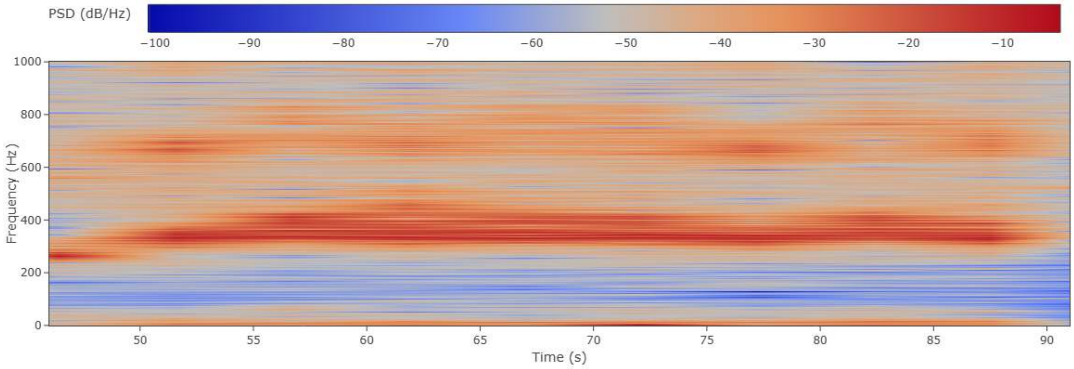
```
32          pt.position.x = msg.position.x
33          pt.position.y = msg.position.y
34          pt.position.z = msg.position.z
35
36          # and velocity
37          pt.velocity.x = msg.velocity.x
38          pt.velocity.y = msg.velocity.y
39          pt.velocity.z = msg.velocity.z
40
41          # yaw control
42          pt.yaw      = msg.yaw
43          pt.yaw_rate = msg.yaw_dot
```

3.3.4 Monitoring

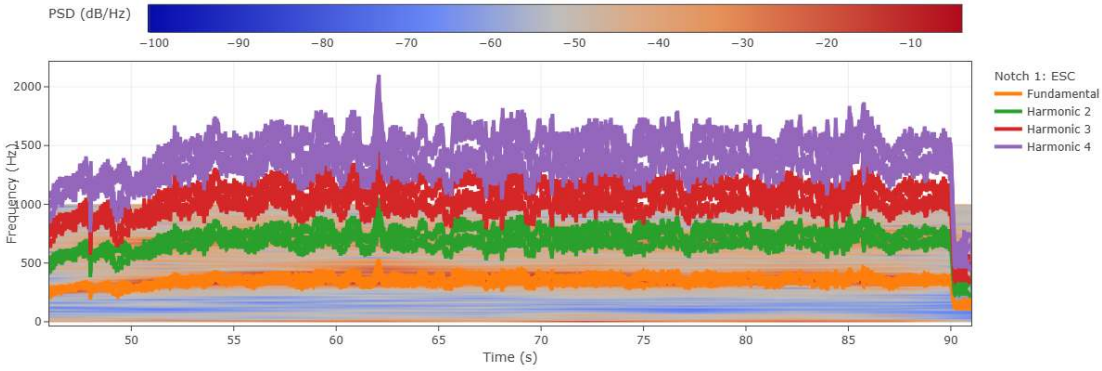
We use Mission Planner [Obo10] on a Windows laptop to monitor the drone's health and flight telemetry. The connected laptop is on the same network as the companion computer. The companion computer forwards MAVLink messages from the flight controller to the ground control station (GCS). Simultaneously, A second laptop running Ubuntu 20.04 and ROS Noetic subscribes to FUEL's mapping and planning topics, visualising the scanning progress in RViz over Wi-Fi and providing us with real-time feedback on navigation and reconstruction. This is not crucial for the intended use case, but it is very helpful as it enables efficient troubleshooting.

3.3.5 Gyroscope Noise Filtering

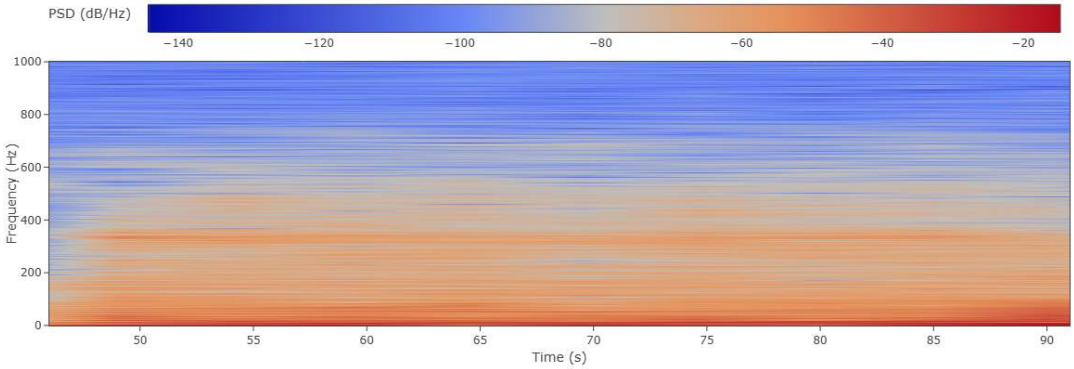
Vibrations transmitted from the spinning motors and propellers cause significant noise in the gyroscope readings on the flight controller. This noise can be problematic as the controller relies on the gyroscope data for stabilization. As these oscillations vary in frequency depending on rotor speed and blade count, they cannot be filtered by a simple fixed notch filter. Instead, the Electronic Speed Controller (ESC) continuously reports the current RPM data of every motor to the Flight Controller. By analyzing these RPM measurements, we can compute the fundamental vibration frequency and its harmonics, then apply notch filters to filter those frequencies out. The harmonics of the base frequency are needed due to the blade count of the propellers. The filtering process can be seen in Figure 3.4. Figure (a) shows the gyroscope noise during a flight without filtering. Figure (b) shows the applied filters that change with the RPM rate of the motors, and Figure (c) shows the filtered gyroscope data with reduced noise.



(a) Gyroscope Noise before Filtering



(b) Filter Frequencies



(c) Gyroscope Noise after Filtering

Figure 3.4: Gyroscope Noise

CHAPTER 4

Results

In this chapter, we review the objectives and the challenges we faced when developing our drone platform. We first examine the design and performance of the flight hardware, and then the autopilot architecture and its integration.

4.1 Hardware

4.1.1 Companion Computer

We powered the Raspberry Pi 5 with a dedicated step-down converter that delivers 5 V at up to 5 A. Without this regulator, the Pi's peripherals would draw too much current, causing signal dropouts. Our initial approach, running both the flight system and the Pi off a single 4S LiPo pack, performed well at rest but suffered voltage sags during takeoff and aggressive maneuvers, which forced the Pi into a shutdown. To eliminate power interruptions, we equipped the Raspberry Pi with its own 2S 720mAh LiPo battery. In practice, this setup delivered about 10–15 minutes of continuous operation with all peripherals and the mapping software running. Unfortunately, there is no battery management for the 2s battery. Therefore, we regularly measured the voltage by hand to avoid damage to the battery cells. When we tested a runtime of 15 minutes per battery, we had no issue with undervoltage.

We used Inkscape [Ink] and a X-Carve 1000 CNC machine [XCa], available to me through my workplace, to design a custom connector plate through several iterative prototypes to securely mount the Raspberry Pi, the step-down converter, and the battery. The plate aligns with the existing frame holes and the Pi's pre-drilled holes to ensure a flexible and lightweight mount.

(a) v1 without stepdown converter

(b) v2 with step-down converter

(c) v3 with space for extra battery

Figure 4.1: Raspberry Pi mount versions

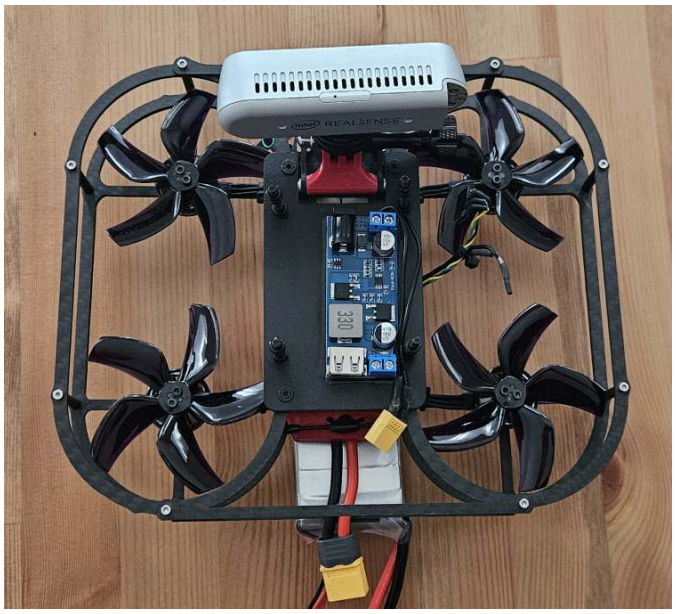


Figure 4.2: Mount fitting on the drone

4.1.2 Calibration

We performed a full visual–inertial calibration of our stereo camera and IMU using the Kalibr framework ([FRS13], [FBS12], [MFS13]), employing a 6×6 AprilTag board as our target. To collect data, we recorded the infrared camera and the IMU data streams while rotating and moving the drone in front of the board. During optimization, Kalibr fitted the camera intrinsics, the stereo extrinsic transform, and the time-offset between camera and IMU, all while minimizing reprojection and inertial residuals.

Our initial attempt used an AprilTag board printed at DIN A4 size on a small desktop

printer. However, the low printing resolution and size led to calibration artifacts and a reprojection error exceeding 2 pixels. To address this, we reprinted the AprilTag grid at DIN A3 size on a professional printer, which produced a higher-quality board. Re-running the same set of calibration trajectories with this improved target reduced the median reprojection error to below 0.2 pixels, as illustrated in Figure 4.3.

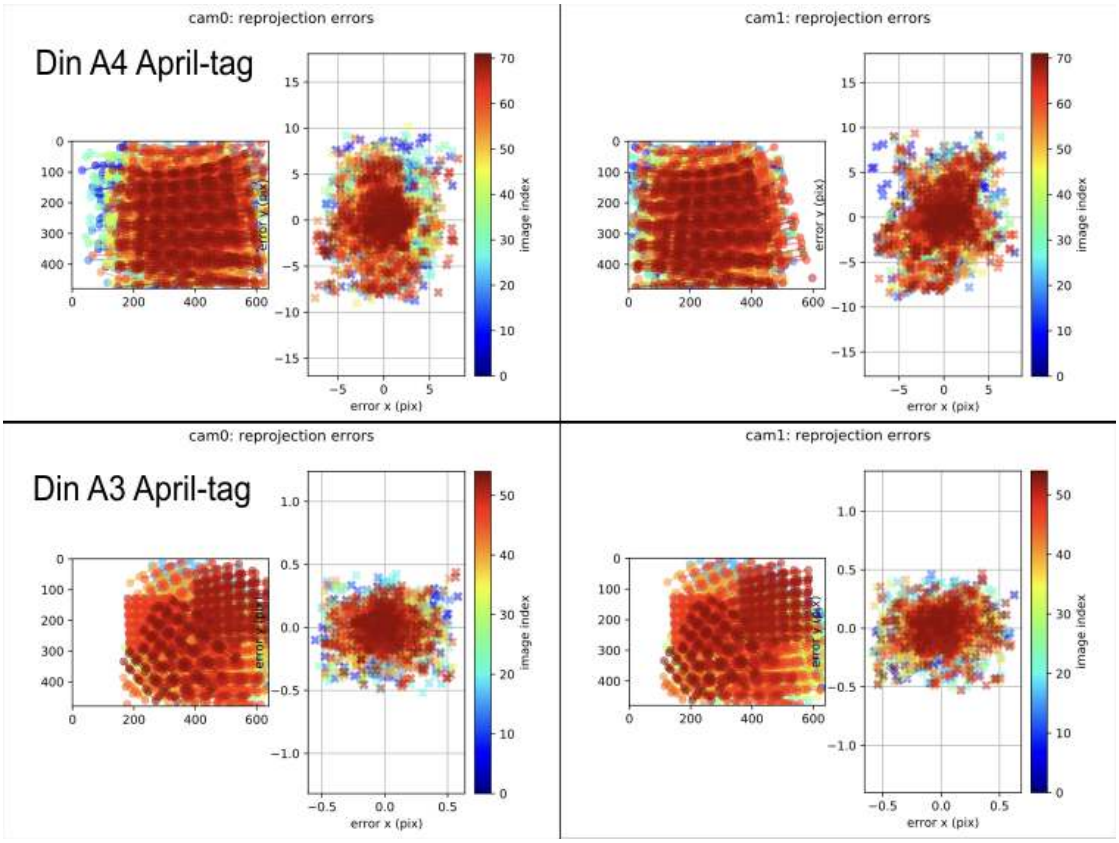


Figure 4.3: Calibration comparison

In addition to spatial calibration, it is essential to characterize the stochastic behavior of the IMU. To this end, we collected 14 hours of stationary IMU data, rigidly mounting the sensor and leaving it motionless in a controlled environment, then processed this log-file with the Allan Variance ROS package [Buc21]. From the resulting Allan variance curves, we extracted key noise parameters for both accelerometer and gyroscope: the noise density/random walk and the bias instability. These values were then incorporated into our EKF settings in Open VINS, ensuring that the inertial data is neither over- nor under-trusted when fusing it with visual measurements.

4.1.3 Weight

Excluding the companion computer and camera, the bare airframe and flight electronics weigh approximately 530 g, leaving headroom for additional hardware. Our chosen scanning payload, a Raspberry Pi 5 paired with an Intel RealSense D435i plus power supply, adds around 230 g, bringing the drone's total takeoff mass to roughly 760 g (Table 4.1).

Component	**Weight (g)**
Sensors + ELRS-Receiver	8
Flight Stack (Flight controller + ESC)	23
720mAh Battery	36
Realsense d435i Camera + cable	77
Motors + Propeller	94
Frame	110
Raspberry Pi 5 + stepdown module	120
3300mAh Battery	292
Total	**760**

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

4.1.4 Flight Time

Before construction, we ran a hover-endurance estimate in Ecalc [Mü04], which predicted 6.7 minutes with our scanning payload. In practice, our real-world hover tests matched this prediction exactly (see Table 4.2). As a separate battery powers the companion computer, its runtime of 10–15 minutes does not limit the drone's flight time.

Setup	**Flight-time (min)**
Basic Flight System	8.5
with Scanning Hardware	6.7

Table 4.2: Flight Time test

4.2 Sensor Integration

We had planned to equip the drone with a number of sensors to increase overall flight safety and stability. Thus, we mounted 2 rangefinders and an optical flow sensor to the drone and used the stereo camera in combination with a small script as a virtual distance sensor. The integration in Ardupilot as proximity sensors was tricky. The upward-facing rangefinder reported 0.0 m whenever it saw no obstacle within its detection range, tricking the controller into thinking the drone was about to collide and causing it to descend or even refuse to lift off in taller rooms, so we ultimately disabled that sensor. In contrast, our downward-facing combo of optical flow and LiDAR improved the hover precision,

reliably holding position within about a 0.5 m radius. Because we did not think of the electrical conductivity of carbon fiber, we smoked one sensor by not isolating it properly.

4.3 Odometry

Our initial odometry pipeline relied on RTAB-Map [LM17], which primarily uses the camera's point-cloud data to estimate the location. In practice, however, fast maneuvers often broke the visual tracking and required manual re-initialization, making it unsuitable for our application.

We thus switched to Open VINS [GEL+20], which fuses stereo imagery with high-frequency IMU readings. This change markedly improved robustness during rapid movements, but we observed occasional "flyaways" where the calculated position sometimes randomly moved with high velocity in one direction. Although our Kalibr-based calibration substantially reduced these errors, they could not be eliminated entirely. This can be seen in Figure 4.4, where in (a) the drone is moved inside a 2m radius,and in (b) the calculated position rapidly moves multiple kilometers away. One reason could be a remaining miscalibration of the camera or, more likely, the IMU. Notably, these errors happened when the drone was in a relatively stationary position.

The Flyaway error led to some major crashes, one of them breaking both rangefinders and the Optical Flow sensor 4.5.

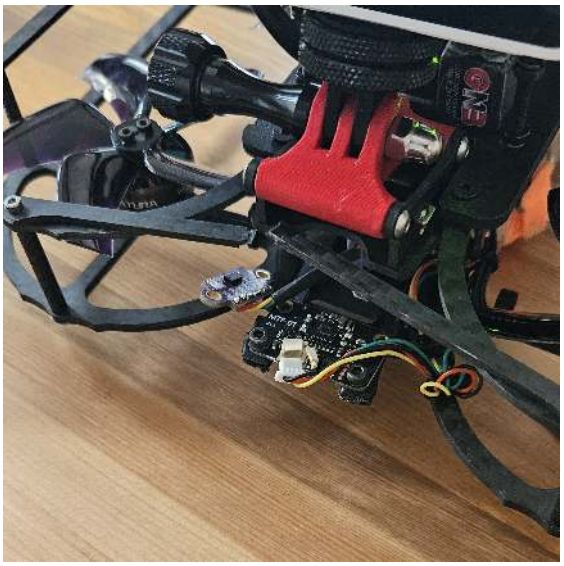
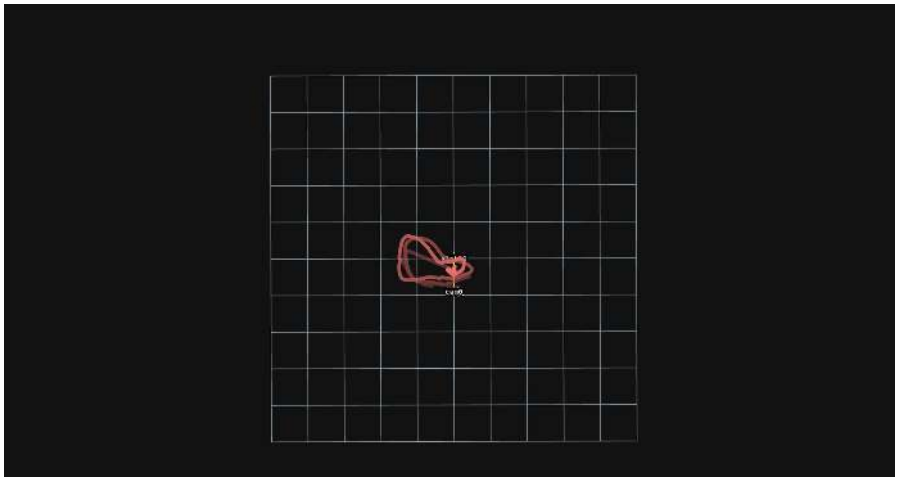
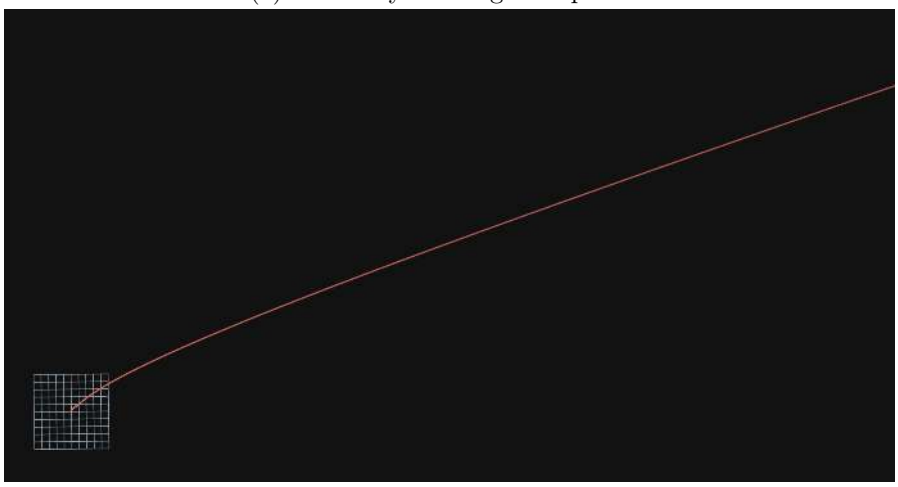


Figure 4.5: Damaged Frame and sensors



(a) odometry working as expected



(b) flyaway error

Figure 4.4: Odometry flyaway

4.4 Mission Planner

Because ELRS supports bidirectional MAVLink since version 3.50, we wanted to use it for monitoring the status of the drone using Mission-Planner [Obo10]. This provides an advantage of a separate data link to monitor the drone, eliminating the need for a companion computer. Unfortunately, when we connected via Mission-Planner to the flight controller with mavros running on the Companion Computer, we experienced slower drone response times, eventually requiring a hard reset of the drone. We suspected loopbacks to be the cause of this slowdown and decreased the risk for such loopbacks by routing the Mavlink messages via the Companion Computer and the already existing Wi-Fi connection to Mission-Planner.

4.5 Scanning

Because we were unable to fully resolve the odometry inconsistencies within the scope of this work, we were unable to execute fully autonomous scanning missions with the drone. Instead, we performed several manual scans with the intended software to test the performance of the Hardware.

4.5.1 FUEL

Our first choice for a mapping and navigation framework was FUEL [ZZCS21]. FUEL seamlessly combines 3D-mapping with Next-Best-View (NBV) planning, building a volumetric occupancy map and computing safe exploration trajectories in one package. Within our ROS Noetic/Ubuntu 20.04 environment, we fed FUEL the real-time odometry from Open VINS and depth measurements from the RealSense D435i. FUEL then published waypoint sequences and flight trajectories over ROS topics. We aimed to manually follow these waypoints to test the mapping and navigation capabilities of the system.

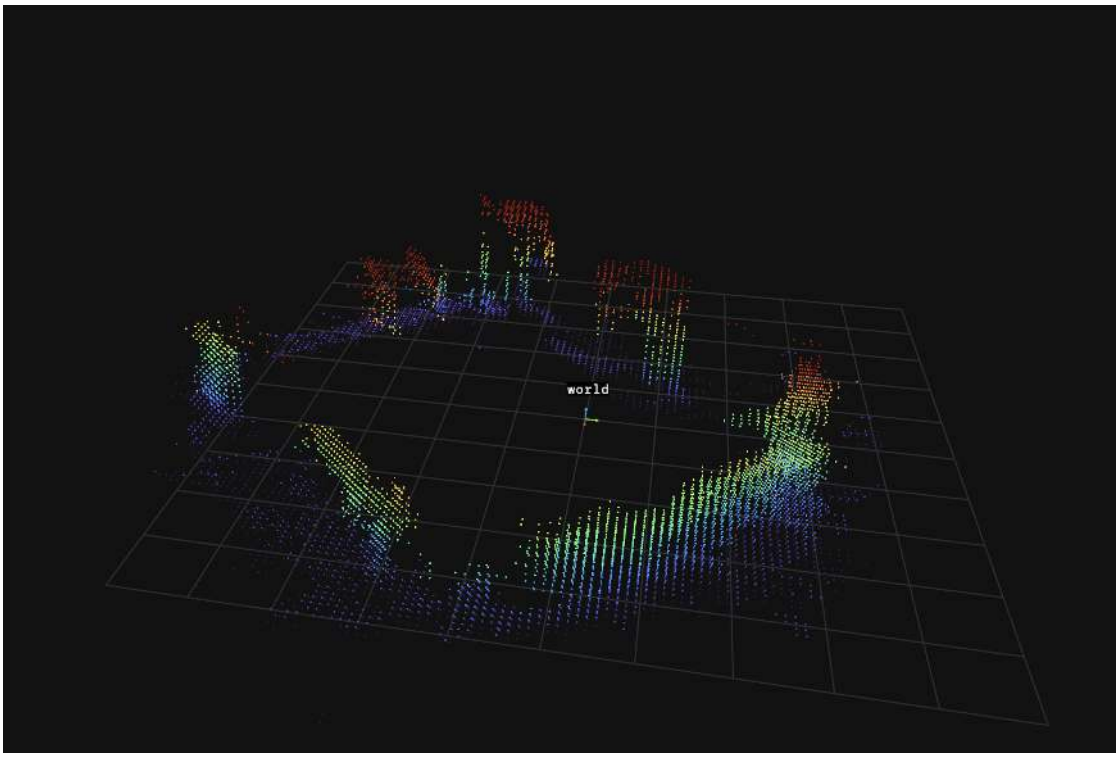


Figure 4.6: The point cloud captured with FUEL

4.5.2 Realityscan

To improve the scan quality of our FUEL-based mapping, we recorded the stereo and the RGB feed during each scanning flight into a rosbag. After the flight, we imported these datasets into Realityscan [Epi]. This handled image alignment, control-point optimization, dense depth-map generation, and mesh reconstruction—streamlining our photogrammetry workflow and delivering detailed, ready-to-use indoor scans without manual intervention. The processing took about 20 min on a dedicated gaming laptop.

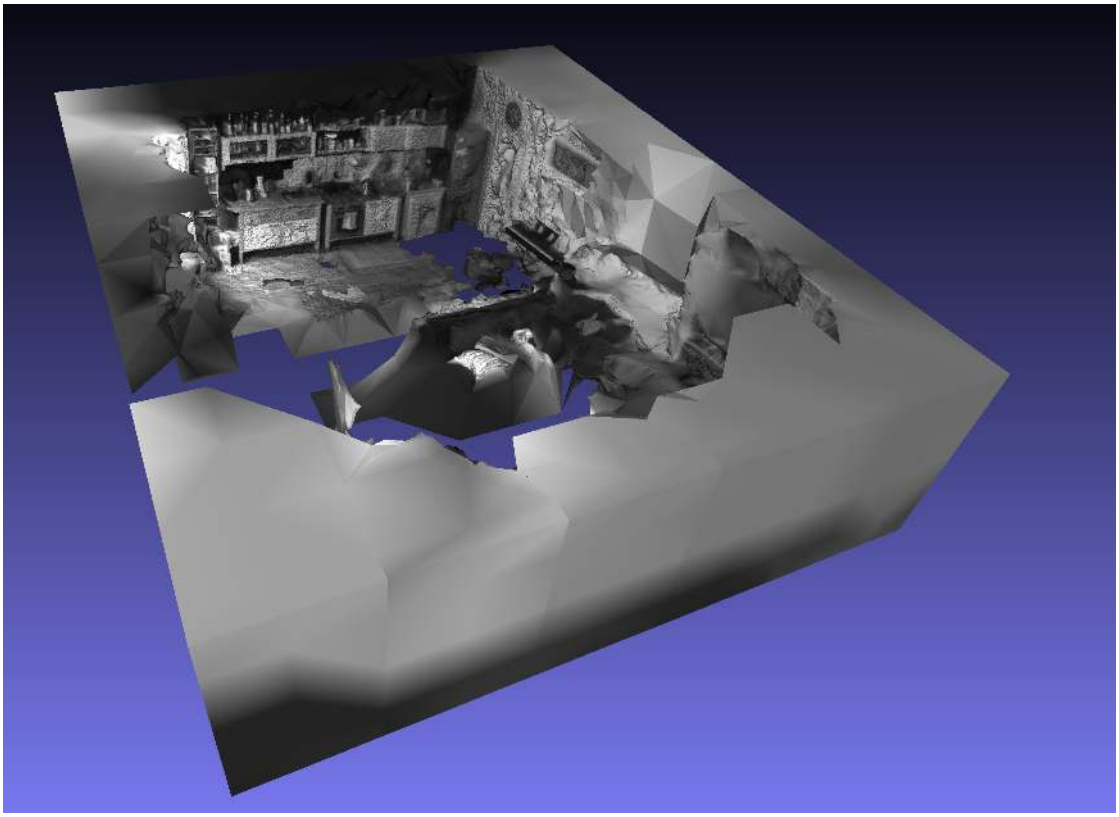


Figure 4.7: The mesh generated with Realityscan

4.5.3 Scan Evaluation

As shown in Figures 4.6 and 4.7, FUEL's real-time scan captures only the room's basic outline with minimal detail, whereas the RealityCapture reconstruction can deliver much finer surface detail, with a few remaining gaps in the mesh. The reason for the low-quality FUEL-scan could be odometry inconsistencies from OpenVINS or the limited computing power of the Raspberry Pi.

CHAPTER 5

Conclusion

With this thesis, we have presented the design, implementation, and evaluation of a compact, flexible drone platform tailored for autonomous indoor 3D scanning. We successfully built the drone platform and were able to fly the drone and 3D scan indoor environments. Given the challenges with odometry stability and system integration, future adaptations are needed for fully autonomous flights. The biggest issue is the occasional "flyaway" in our visual–inertial odometry pipeline. Despite careful calibration and noise characterization, flight maneuvers can still induce spurious jumps in the estimated pose. Resolving this will likely require tighter hardware timestamps, more sophisticated outlier rejection in the EKF, or the addition of an overall more capable positioning system.

Regardless of these limitations, we presented a platform that demonstrates the feasibility of a fully self-contained, server-free indoor scanning UAV. Its modular mechanical and electrical design proved robust under manual flights, and our ROS-based software stack allowed rapid integration of perception and control modules.

5.1 Future Work

The Platform could be improved in a few key areas to make it fully autonomous and more reliable. Addressing the residual odometry "flyaways" will be essential. This could be achieved with a different odometry approach, either with other algorithms or different hardware solutions like dedicated pose-estimation systems.

The used Realsense d435i camera can provide decent depth images for 3d mapping but relies on an integrated IR projector that had to be switched off because of interference with the visual odometry system. To avoid this compromise, we propose decoupling pose estimation from mapping by using two dedicated sensors: one optimized for robust odometry and another one dedicated solely to depth capture. By assigning each task to

the sensor best suited for it, they can operate at peak performance without interference and compromises. To reduce the risk of interference, the sensors could use different technologies, such as LiDAR and visual imaging, or an IR filter could be applied to block the projected pattern from one of the sensors.

The current power solution, while effective at preventing Pi drop-outs, relies on manual battery monitoring. Integrating an onboard power-management system with under-voltage cutoff and charge estimation would enable longer missions without human intervention.

Overview of Generative AI Tools Used

During the preparation of this thesis, I leveraged AI tools to streamline research and enhance clarity. Google Gemini was used for literature searches and summarization, while ChatGPT (o4 and 4.5-beta) assisted with ROS and ArduPilot troubleshooting, as well as refining formulations and improving overall readability. I also used Grammarly for Spelling correction.

List of Figures

List of Tables

List of Algorithms

Listings

Bibliography

[B] Boris B. Betaflight - Pushing the Limits of UAV Performance | Betaflight — betaflight.com. https://betaflight.com/. [Accessed 03-08-2025].

[Buc21] Russell Buchanan. Allan variance ros, November 2021.

[C] Open Source Contributor
and The Community. GitHub - iNavFlight/inav: INAV: Navigation-enabled flight control software — github.com. https://github.com/iNavFlight/inav.git. [Accessed 03-08-2025].

[CBH+24] Hamish A. Campbell, Vanya Bosiocic, Aliesha Hvala, Mark Brady, Mariana A. Campbell, Kade Skelton, and Osmar J. Luiz. Emerging research topics in drone healthcare delivery. *Drones*, 8(6), 2024.

[Com09] Jordi Muñoz + Community. ArduPilot — ardupilot.org. https://ardupilot.org/, 2009. [Accessed 20-07-2025].

[CSSC+22] Amalia Lelia Crețu-Sîrcu, Henrik Schiøler, Jens Peter Cederholm, Ion Sîrcu, Allan Schjørring, Ignacio Rodriguez Larrad, Gilberto Berardinelli, and Ole Madsen. Evaluation and comparison of ultrasonic and uwb technology for indoor localization in an industrial environment. *Sensors*, 22(8), 2022.

[DPH+18] Lachlan Dowling, Tomas Poblete, Isaac Hook, Hao Tang, Ying Tan, Will Glenn, and Ranjith R Unnithan. Accurate indoor mapping using an autonomous unmanned aerial vehicle (uav), 2018.

[EHa] EHang. Dronepixel | Drohnen Lichtshow — drone-pixel.com. https://www.drone-pixel.com/. [Accessed 03-08-2025].

[Epi] EpicGames. RealityScan | 3D models from images and laser scans — realityscan.com. https://www.realityscan.com/. [Accessed 03-08-2025].

[FBS12] Paul Furgale, Timothy Barfoot, and Gabe Sibley. Continuous-time batch estimation using temporal basis functions. *The International Journal of Robotics Research*, 34:2088–2095, 05 2012.

[FRS13] Paul Furgale, Joern Rehder, and Roland Siegwart. Unified temporal and spatial calibration for multi-sensor systems. *Proceedings of the ... IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1280–1286, 11 2013.

[GEL+20] Patrick Geneva, Kevin Eckenhoff, Woosik Lee, Yulin Yang, and Guoquan Huang. OpenVINS: A research platform for visual-inertial estimation. In *Proc. of the IEEE International Conference on Robotics and Automation*, Paris, France, 2020.

[HAJ+21] Ramyad Hadidi, Bahar Asgari, Sam Jijina, Adriana Amyette, Nima Shoghi, and Hyesoon Kim. Quantifying the design-space tradeoffs in autonomous drones. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21, page 661–673, New York, NY, USA, 2021. Association for Computing Machinery.

[Ink] Inkscape Project. Inkscape.

[Int] Intel. Intel® RealSense™ Tiefenkamera D435i – Produktspezifikationen | Intel — intel.de. `https://www.intel.de/content/www/de/de/products/sku/190004/intel-realsense-depth-camera-d435i/specifications.html`. [Accessed 08-09-2025].

[JNS+24] Nusrat Jahan, Tawab Bin Maleque Niloy, Jannatul Fahima Silvi, Mahdi Hasan, Ishrat Jahan Nashia, and Riasat Khan. Development of an iot-based firefighting drone for enhanced safety and efficiency in fire suppression. *Measurement and Control*, 57(10):1464–1479, 2024.

[Kal60] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, March 1960.

[LM17] Mathieu Labbé and François Michaud. Long-term online multi-session graph-based splam with memory management. *Autonomous Robots*, 42(6):1133–1150, November 2017.

[Low99] D.G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, page 1150–1157 vol.2. IEEE, 1999.

[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), 2022.

[MFS13] Jérôme Maye, Paul Furgale, and Roland Siegwart. Self-supervised calibration for robotic systems. *IEEE Intelligent Vehicles Symposium, Proceedings*, 06 2013.

[MG62] Smith Mc Gee, Schmidt. Application of statistical filter theory to the optimal estimation of position and velocity on board a circumlunar vehicle. *nasa_techdocs*, 1962.

[Mü04] Markus Müller. eCalc - reliable electric drive simulations — ecalc.ch. `https://www.ecalc.ch/`, 2004. [Accessed 04-08-2025].

[Obo10] Michael Oborne. GitHub - ArduPilot/MissionPlanner: Mission Planner Ground Control Station for ArduPilot — github.com. `https://github.com/ArduPilot/MissionPlanner.git`, 2010. [Accessed 04-08-2025].

[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 3.2, page 5. Kobe, 2009.

[RNS+16] Joern Rehder, Janosch Nikolic, Thomas Schneider, Timo Hinzmann, and Roland Siegwart. Extending kalibr: Calibrating the extrinsics of multiple imus and of individual axes. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4304–4311, 2016.

[RRKB11] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, page 2564–2571. IEEE, November 2011.

[TFG+23] Murad Tukan, Fares Fares, Yotam Grufinkle, Ido Talmor, Loay Mualem, Vladimir Braverman, and Dan Feldman. Orbslam3-enhanced autonomous toy drones: Pioneering indoor exploration, 2023.

[XCa] X-Carve Instructions: Welcome! — x-carve-instructions.inventables.com. `https://x-carve-instructions.inventables.com/1000mm/`. [Accessed 08-09-2025].

[Zha00] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

[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.

[ZZCS21] Boyu Zhou, Yichen Zhang, Xinyi Chen, and Shaojie Shen. Fuel: Fast uav exploration using incremental frontier structure and hierarchical planning. *IEEE Robotics and Automation Letters*, 6(2):779–786, 2021.