Politecnico di Torino

Politecnico di Torino

Master's Degree in Mechatronic Engineering
Academic Year 2025/2026

Graduation Session March 2026

Configuration and Experimental Validation of PX4 UAVs with VICON-Based Indoor Localization

Supervisor:	Co-Supervisors:	Candidate:
Prof. Alessandro Rizzo	Dr. Stefano Primatesta	Giorgio Capotorto
	Dr. Andrea Usai	

Abstract

This thesis presents the system-level configuration and validation of PX4-based Unmanned Aerial Vehicles integrated with a VICON motion capture system within the indoor flight arena facility of Politecnico di Torino. The work focuses on enabling reliable indoor localization and autonomous flight through the correct integration of external motion-capture data into the onboard estimation and control architecture on the PX4 autopilot. The developed setup involves multiple UAV platforms equipped with PX4 autopilot and companion computers running ROS 2. Motion-capture pose measurements provided by the VICON system are distributed over the local network and processed on the companion computer before being delivered to the flight controller as external odometry. The onboard EKF2 estimator fuses inertial measurements from the IMU with VICON-based pose information through an Extended Kalman Filter, producing a consistent and unified state estimate used for flight control and navigation. Particular attention is devoted to practical integration challenges, including reference-frame alignment, timestamp semantics, LAN-based clock synchronization, communication reliability between the companion computer and the flight controller, and the overall system stability. A local time-synchronization strategy based on Chrony is implemented, and the motion-capture data pipeline is experimentally characterized to quantify latency and ensure correct estimator delay compensation. The configured system is validated through indoor flight experiments, including autonomous missions executed in offboard mode via a Python-based planner. A repeatable test sequence consisting of takeoff, trajectory tracking, and safe landing is used to evaluate stability and estimation consistency. The resulting configuration provides a reproducible baseline for future multi-UAV experiments and coordinated flight tests within the facility.

Acknowledgements

To Dr. Stefano Primatesta: thank you for guiding me throughout this project, for your trust, for giving me this opportunity, and for supporting me every step of the way. Thank you for being truly, and I mean truly, always present and available. But above all, thank you for the person you are, for bringing a sense of humanity back into this environment, and for leaving a deep mark on me.

To my mother and father: words can only say so much. You have been my foundations, my home even from 1033 km away. Thank you for helping me get back on my feet after every fall. None of this would have happened without you, without your belief in me from the very first day and your encouragement from the very first moment. That decision to "send me away" was partly by chance, partly a challenge, and partly pure madness... and for all of this, I will always be grateful to you, with all my love.

To my girlfriend: thank you for standing by my side, for repeating the same words over and over, just to convince me that I was good enough, that I could do it, and that I should never give up, and for loving me through it all.

To my closest relatives: thank you for helping me grow, improve, and for truly caring about me. To those who have been like parents to me, and to the one who has been like a brother and a best friend, thank you for always being there with love, guidance, and unwavering support.

To my dearest friends: to the real ones, each of you knows how present, important, and impactful you have truly been in my life. Thank you for being there for me throughout all these years, through problems, madness, training sessions, adventures... and more problems. Thank you for being here. I love you all.

And finally, I thank myself. For always getting back up, for turning my life upside down, for going against everything and everyone, and for making it through this long and difficult journey "without hope." For never giving up, and for always staying true to who I am. Keep pushing.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Achieving reliable indoor autonomy for unmanned aerial vehicles (UAVs) is primarily a system-integration and validation problem. Motion-capture systems can provide high-accuracy pose measurements, yet autonomous flight performance depends on whether the entire pipeline—from external localization to onboard state estimation and finally to the control interface—is consistent, time-aligned, and reproducible across experimental sessions.

In practice, the challenge is not the lack of information, but the risk of introducing subtle inconsistencies when connecting heterogeneous components: a motion-capture workstation, a LAN transport layer, a companion computer running ROS 2, and an autopilot with strict expectations on frames and timestamps. Small mismatches in coordinate-frame conventions, timestamp handling, or communication settings can lead to estimator inconsistency, incorrect yaw/heading behaviour, and ultimately degraded stability and trajectory tracking. These issues are often difficult to diagnose because the system may appear to operate correctly at the topic level while still fusing measurements in the wrong frame or at the wrong effective delay.

For this reason, indoor autonomy must be approached with an engineering mindset focused on observability and repeatability. The goal is not only to "make the drone fly" once, but to build a workflow that can be restarted after reboots and configuration changes, and that provides clear evidence to validate each stage of the chain (raw motion-capture pose, converted external odometry, estimator fusion, and control response). This thesis follows that direction by emphasizing staged verification, coherent time bases, and practical diagnostics (e.g., yaw consistency checks), so that experimental results can be reproduced and extended with confidence.

The Politecnico di Torino aerial robotics test facility offers a controlled environment equipped with a Vicon motion-capture system, allowing for safe and repeatable experiments. Such a facility is valuable not only to demonstrate indoor flight, but also to experimentally validate the software architecture needed to integrate external localisation into a modern autopilot stack and to execute autonomous behaviours through offboard control.

1.2 Objectives

The overall objective of this thesis is to design, implement, and experimentally validate an end-to-end architecture that enables a PX4-based quadrotor, equipped with a companion computer running ROS 2, to operate indoors using Vicon-based localization and to execute autonomous flight tasks in offboard mode.

This objective can be decomposed into three engineering goals:

- **External localization integration:** transform and deliver motion-capture measurements as PX4-compatible external odometry, ensuring correct frame conventions and orientation representation.
- **Timing consistency and delay handling:** establish clock synchronization in a local network and characterize end-to-end delays to support reliable fusion within the PX4 EKF2 estimator.
- **Autonomous flight validation:** implement an offboard control pipeline in Python and demonstrate repeatable autonomous indoor flights consisting of takeoff, short trajectory execution, and safe landing inside the indoor facility.

1.3 Methodology

The work follows an incremental methodology aimed at reducing integration risk and improving reproducibility. First, a stable software baseline is established in simulation using PX4 SITL with Gazebo. This step is used to validate the development environment, confirm correct communication between PX4 and ROS 2, and provide a controlled debugging workflow.

Second, the same architecture is transferred to the real platform and integrated with the Vicon infrastructure. At this stage, special attention is devoted to reference frames and timing. Motion-capture data are accessed through the facility network and processed by ROS 2 nodes on the companion computer, which convert pose measurements into a PX4-compatible external odometry representation. Since EKF2 compensates for sensor delays internally, clock synchronization and latency characterization are treated as first-class requirements of the integration.

Finally, autonomous flight experiments are carried out using a Python offboard controller. The controller executes a complete indoor mission consisting of takeoff, short trajectory execution, and a safe landing procedure. This mission is used as a repeatable experimental benchmark to evaluate the full pipeline, from localization and estimation to closed-loop control execution.

1.4 Contributions

The main contributions of this thesis are:

- a reproducible software setup based on Ubuntu and ROS 2 Jazzy, integrated with PX4 Autopilot and a Micro XRCE-DDS bridge enabling systematic access to PX4 topics from ROS 2 and the injection of external odometry;
- a real-world integration pipeline for Vicon-based localization, including explicit handling of coordinate-frame conventions and an engineering workflow for time synchronization and delay characterization in a LAN environment;
- an offboard control implementation in Python and an experimental validation protocol based on a repeatable indoor mission (takeoff, short trajectory execution, and landing), providing a practical benchmark for evaluating the system behaviour.

1.5 Thesis Structure

The remainder of this thesis is organized as follows. Chapter 2 presents the UAV platform and the indoor test facility. Chapter 3 describes the Vicon motion-capture system and the data distribution pipeline used to access tracking measurements. Chapter 4 details the software architecture based on PX4, ROS 2, and the Micro XRCE-DDS bridge, and explains the end-to-end data flow from motion capture to onboard estimation. Chapter 5 introduces the simulation baseline and the validation steps performed in SITL. Chapter 6 discusses the real-world integration, including frame transformations, timing strategy, and estimator configuration. Chapter 7 reports the offboard control design and the experimental results obtained during indoor flights. Finally, Chapter 8 summarizes conclusions, limitations, and future work.

Chapter 2

Platform and Test Facility

Figure 2.1: PX4 Vision Autonomy Development Kit

2.1 System Overview

The experimental setup considered in this thesis consists of three main layers: (i) an onboard flight-control stack based on PX4, (ii) an onboard companion computer running ROS 2 for integration and autonomy features, and (iii) an indoor test facility equipped with a Vicon motion-capture system. From a system-integration perspective, this separation is essential: the flight controller executes real-time

estimation and control, whereas the companion computer hosts higher-level software components (middleware, logging, data conversion, and offboard control) that are easier to develop and iterate.

The reference hardware platform adopted in the laboratory is the *PX4 Vision Autonomy Development Kit*, which integrates a Pixhawk-class flight controller, a companion computer, and a standard sensing payload typically used in autonomy research (depth camera, optical flow, and distance sensing). The kit is commonly used as a baseline for indoor autonomy and facility integration. [1]

From a hardware and software co-design standpoint, the key idea is the separation between:

- the **Flight Management Unit (FMU)**, which runs safety-critical real-time estimation and control loops; and
- the **Companion computer**, which runs Linux and hosts the integration layer and high-level autonomy (ROS 2 nodes, logging, conversions, and offboard control).

This architecture is widely adopted in robotics because it enables rapid iteration on the companion side while keeping the real-time flight stack isolated and robust. A practical advantage of this two-layer architecture is that it supports a clean experimental workflow. The flight controller can remain configured in a conservative and stable way (EKF2, flight modes, safety logic), while development iterations mostly happen on the companion computer (ROS 2 nodes, conversion logic, and logging). This separation reduces the risk of introducing regressions in the real-time flight stack and makes troubleshooting more systematic, because issues can be localized either to the external localization pipeline or to the onboard estimation/control layer. In the remainder of the thesis, this platform is used as a reference baseline to study how motion-capture measurements can be injected into PX4 as external odometry for reliable indoor autonomy.

2.2 Onboard Architecture

2.2.1 Flight Controller

Figure 2.2: Pixhawk 6C flight controller

The flight controller [2] runs PX4 Autopilot and is responsible for low-level stabilization, flight-mode management, and onboard state estimation through the EKF2 estimator. In the context of indoor autonomy, EKF2 plays a central role because it fuses inertial measurements with additional aiding sources. Since GNSS is not available in the indoor test facility, the primary aiding source used in this thesis is motion-capture-based external odometry.

A key engineering implication is that the flight controller is sensitive not only to the numerical values of external measurements, but also to *how* they are represented and *when* they are delivered. For this reason, frame conventions (world and body axes, handedness, quaternion conventions) and timestamp semantics are treated as first-class integration requirements throughout the thesis. When any of these elements is inconsistent, EKF2 may fuse incorrect information or reject the aiding source altogether, leading to unstable or non-repeatable behavior once offboard control is enabled.

The experiments presented in this work are conducted with a Pixhawk 6C-class controller. Within the overall architecture, it executes PX4 in real time and provides the estimator and control interfaces required for indoor flight, while exposing the integration points needed to inject external odometry and accept offboard setpoints.

2.2.2 Companion Computer

Figure 2.3: Companion computer used for onboard robotics integration

The companion computer hosts the ROS 2 ecosystem and runs the software components that interface the UAV with the external infrastructure (Vicon/VRPN) and with ground-station tools. In the reference platform adopted in the laboratory, the companion computer is integrated within the vehicle architecture; Figure 2.3 is shown as an illustrative example to convey the concept of a companion computer and the typical form factor of embedded Linux boards used for robotics integration. [1]

In this thesis, the companion computer is the main execution target for:

- the ROS 2 middleware and the nodes that receive and process motion-capture data;
- the Micro XRCE-DDS setup used to bridge selected PX4 topics into the ROS 2 ecosystem;
- the Python offboard controller used for autonomous indoor missions;
- data collection and debugging tools (e.g., PlotJuggler and rosbag).

This split is practical during experimental work: the companion computer provides a flexible environment to iterate on integration logic (frame conversion, timestamp handling, launch automation), while the flight controller remains dedicated to safety-critical estimation and control loops. As a result, most integration changes can be implemented and tested without modifying the core PX4 configuration, which helps keep the workflow repeatable across sessions.

2.3 Onboard Sensing Payload

The platform includes a sensing payload typically adopted for indoor navigation and research prototyping. In this work, the primary localization reference is provided by the external motion-capture system; nevertheless, onboard sensors remain relevant for completeness and for describing the baseline hardware configuration.

2.3.1 Stereo/Depth Camera

The kit includes a front-mounted depth sensing module, commonly used in autonomy research for perception tasks (depth maps, visual features, and obstacle awareness). In a motion-capture-based workflow, the camera is not required for localization, but documenting its presence is useful for two reasons: (i) it clarifies the reference hardware configuration used in the lab, and (ii) it provides a natural path for future extensions toward onboard perception-based navigation. [1]

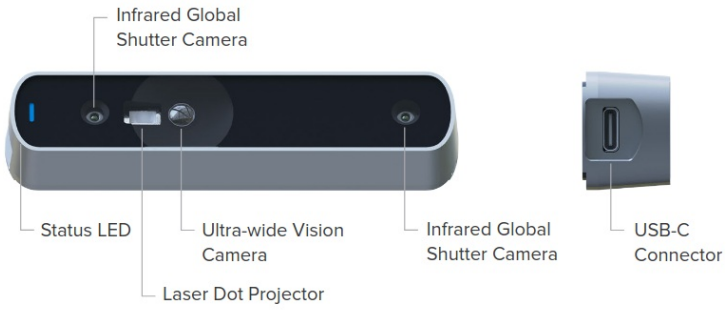

Figure 2.4: Onboard RGB-D/stereo camera module

2.3.2 Optical Flow

The platform includes a downward-facing optical flow sensor, which estimates planar motion relative to the ground by tracking image texture across consecutive frames. In PX4, optical flow can be used as a velocity aiding source in GNSS-denied environments and is particularly useful for indoor flight at low altitude, where sufficient ground texture is available. [3, 4]

In this thesis, optical flow is part of the baseline hardware configuration; however, the primary aiding source for EKF2 during the reported experiments is the Vicon-based external odometry. Optical flow is therefore treated as an available onboard sensing resource rather than the main localization reference.

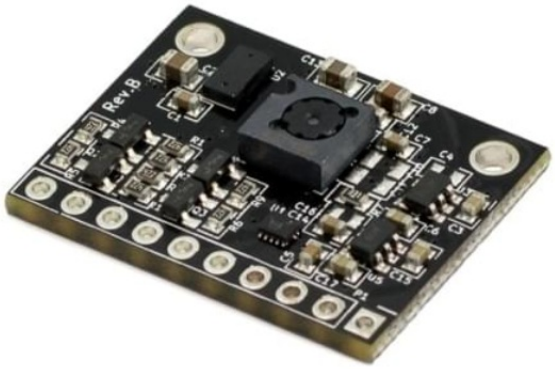

Figure 2.5: Optical flow sensor module

2.3.3 Distance Sensing

A short-range distance (rangefinder) sensor is also included in the reference payload. Range measurements are typically used to estimate the height above ground and, when combined with optical flow, to provide metric scaling of planar velocity estimates. In PX4, distance sensing can also support altitude holding and improve robustness close to the ground, complementing barometric altitude which may be less reliable in fast indoor maneuvers. [5]

As for optical flow, distance sensing is not used as the primary localization source in this thesis; the experimental flights rely on Vicon-based external localization for estimator fusion. Nevertheless, documenting the presence of the range sensor is important because it is part of the standard kit configuration and enables future extensions toward more autonomous onboard navigation.

Figure 2.6: Distance (range) sensor module

2.3.4 GNSS Module (Reference Kit Component)

Even if GNSS is not available indoors and is not used for the experimental flights presented in this thesis, the platform includes a GNSS module as part of the standard kit configuration. Including this component in the platform description clarifies that the indoor experiments intentionally rely on external localization rather than satellite-based navigation. [1]

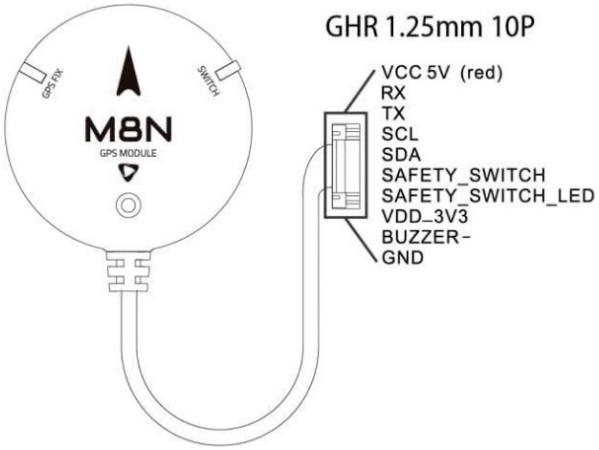

Figure 2.7: GNSS module included in the reference kit

2.4 Power Distribution and Wiring

2.4.1 Power Module and Supply Path

A dedicated power module is used to provide regulated power to the avionics and to measure battery voltage/current for monitoring. The reference platform uses a PM07 power module, which supports multi-cell LiPo inputs (2S–14S) and provides a regulated 5.2 V output (up to 3 A) for the onboard electronics, while exposing voltage and current sensing signals to the flight controller for power monitoring and safety logic. [6]

2.4.2 Reference Wiring Layout of the Kit

For completeness, Figure 2.10 summarizes the typical wiring layout of the PX4 Vision kit, including avionics power, sensor connections, and the companion computer interfaces. This high-level view is useful during integration because

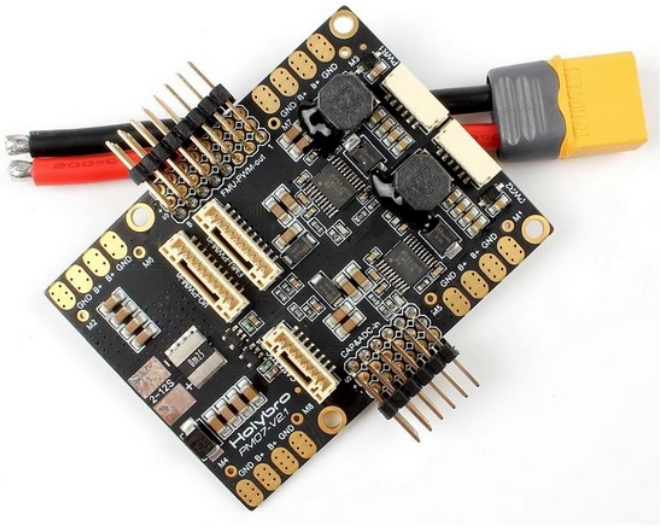

Figure 2.8: Power module (PM07)

it clarifies which signals belong to the real-time flight stack (FMU ports) and which belong to the companion-computing and payload side. [1]

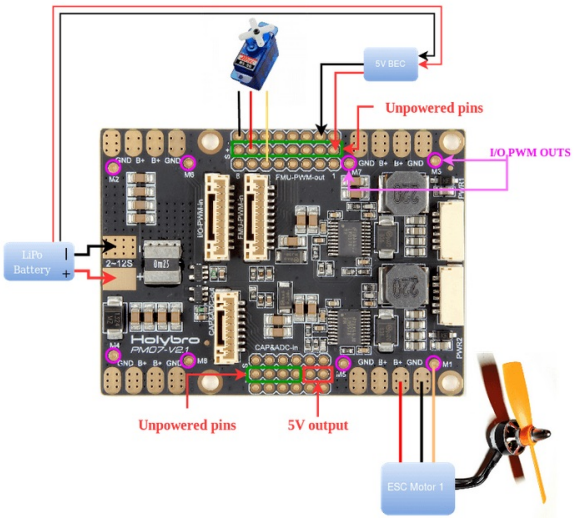

Figure 2.9: Example wiring diagram of the PM07 power module

Figure 2.10 provides a compact but very informative overview of how power and data connections are organized in the PX4 Vision kit. Beyond being a simple assembly reference, it highlights the practical separation between the flight-critical avionics managed by the flight controller and the higher-level integration functions executed on the companion computer. This separation is central to the experimental workflow adopted in this thesis, because it makes the overall system both modular and easier to debug.

From a power-distribution perspective, the diagram emphasizes the role of the power module as the entry point of the supply path: it delivers a regulated rail to the avionics and forwards battery voltage/current measurements to the autopilot. This aspect is relevant in practice because repeatability does not only depend on software: a stable boot sequence and a clean power path reduce intermitteAQnt failures (e.g., peripherals that reboot, sensors that appear/disappear, or inconsistent start-up behavior across sessions). In other words, the same software stack tends to behave more predictably when the underlying hardware initialization is deterministic.

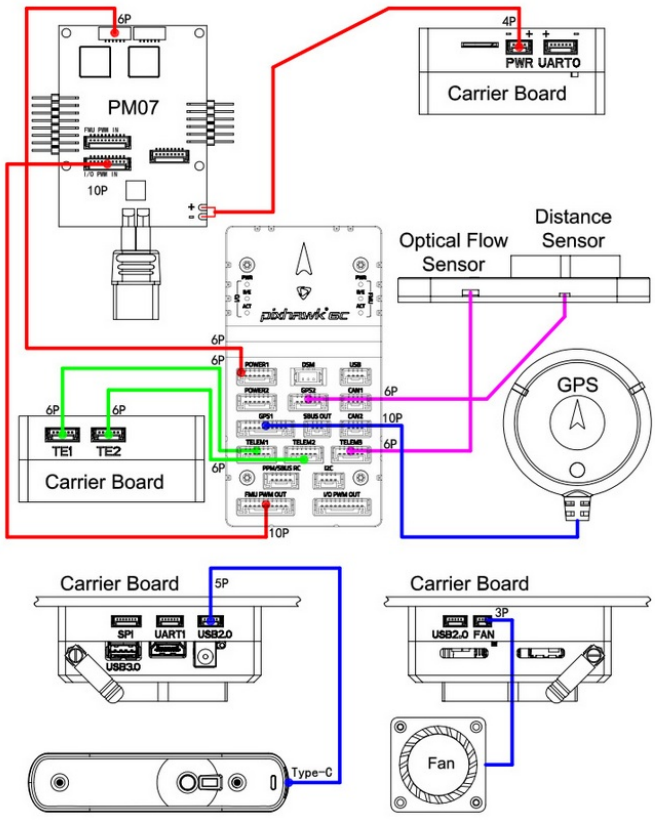

Figure 2.10: High-level wiring overview of the PX4 Vision kit

On the data side, the figure clarifies how the carrier board and the standard connectors group signals according to their function. Interfaces such as TELEM/UART and related buses provide the channel through which the companion computer can exchange information with PX4 (telemetry monitoring, state inspection, and control interfaces). The same wiring layout also shows how optional payload sensors are connected as part of the reference kit configuration. While the experimental flights presented in this thesis primarily rely on motion capture as the external localization source, documenting the standard wiring is still valuable: it defines the baseline hardware platform used in the laboratory and makes explicit which interfaces remain available for future extensions (for example, adding onboard aiding sensors or additional payloads).

Overall, the wiring overview reinforces a key idea behind the platform choice: the kit provides a structured and modular architecture, where a dedicated flight controller executes real-time estimation and control, and a Linux-based companion computer hosts integration and autonomy logic. This organization is well suited to indoor experimentation because it supports both safety (through a robust flight stack) and flexibility (through rapid iteration on the companion side).

2.5 Networking and Ground Station Setup

Indoor experiments require a reliable local network to connect the Vicon infrastructure, the companion computer, and the development workstation. A practical constraint of the laboratory environment is the lack of Internet access on some machines. In the working configuration adopted in this thesis:

- the Vicon workstation runs Windows and operates without Internet access;
- the UAV companion computer runs Ubuntu 24.04 and also operates without Internet access;
- a personal laptop running Ubuntu (with Internet access) is used for development tasks, package preparation, and data transfer when needed.

This setting motivated a reproducible workflow for software deployment, logging, and time synchronization over LAN, which becomes particularly relevant when motion-capture timestamps must be aligned with onboard estimation.

For configuration and monitoring, QGroundControl [7] is used as ground-station software to inspect PX4 status, modify parameters, and verify estimator behaviour. In addition, ROS 2 tools and logging utilities (e.g., rosbag) are used for validation and offline analysis.

2.6 Indoor Test Facility

The experiments are performed inside the Politecnico di Torino indoor flight cage equipped with a Vicon motion-capture system. The facility provides a controlled and safe environment, enabling repeatable tests and reducing the influence of external disturbances typically present in outdoor experiments. Motion capture supplies high-rate pose measurements that can be leveraged as external odometry for indoor navigation and autonomous flight.

The facility layout and the motion-capture pipeline are described in detail in Chapter 3. Here, it is important to highlight the primary constraint that drives the overall design: since GNSS is not available indoors, reliable autonomy depends on the correct integration of Vicon-based localization into the PX4 estimation and control loops.

2.7 High-Level Experimental Workflow

To ensure reproducibility, the experimental workflow adopted in this thesis follows a structured sequence:

1. establish connectivity and verify that motion-capture data are available on the companion computer as ROS 2 topics;
2. validate the PX4–ROS 2 interface and confirm that external odometry can be delivered to PX4 in a consistent representation;
3. execute autonomous indoor tests using the Python offboard controller, using a repeatable mission profile consisting of takeoff, short trajectory execution, and safe landing as benchmark.

This workflow provides a clear separation between infrastructure validation and autonomy validation, and it supports systematic debugging when discrepancies emerge (e.g., frame inconsistencies or timing misalignment).

2.8 Summary

This chapter introduced the reference platform and the indoor test environment used throughout the thesis. The PX4 Vision Autonomy Development Kit provides a modular architecture based on a Pixhawk-class flight controller for real-time estimation and control, and a companion computer for ROS 2 integration, logging, and offboard autonomy. The onboard sensing payload and the standard wiring layout were presented to document the baseline hardware configuration adopted in the laboratory.

The chapter also described the practical experimental context: a LAN-only networking setup, ground-station monitoring through QGroundControl, and an indoor flight cage equipped with Vicon motion capture. These elements define the boundary conditions that drive the integration choices of the following chapters, where the focus moves from hardware description to the software pipeline that delivers motion-capture pose information to PX4 as external odometry and enables repeatable indoor autonomous flights.

Chapter 3

Indoor Facility and Vicon System

3.1 Indoor Flight Cage and Motion Capture Infrastructure

All experimental activities of this thesis are conducted inside the Politecnico di Torino indoor flight cage equipped with a Vicon motion-capture system. The facility provides a controlled and safe environment where experiments can be repeated under comparable conditions. This is a key requirement for validating an autonomy pipeline, since outdoor tests are often affected by weather variability, regulatory constraints, and limited repeatability.

In the laboratory, the Vicon system is already deployed and operational. A dedicated workstation runs the Vicon software and maintains a digital representation of the capture volume, including the tracked rigid bodies. In this configuration, the UAV is detected within the arena and its pose is estimated in real time, providing position and orientation at high update rate [8].

Besides safety, the main advantage of the cage is methodological: it enables a structured validation workflow where changes to the software stack can be tested in a repeatable way. This is particularly relevant for motion-capture-based autonomy, where integration issues are often subtle (frame conventions, yaw sign, timestamp semantics) and require multiple iterations to isolate and fix.

3.2 Rigid-Body Tracking and Pose Estimation

Vicon pose estimation is based on the detection of reflective markers by a calibrated multi-camera setup. Markers arranged on the vehicle define a rigid body, and the tracking software estimates its full six-degree-of-freedom state (3D position and 3D orientation) in the global reference frame of the capture volume [8].

The availability of a real-time 6-DoF pose enables two crucial capabilities for this thesis:

- **Ground-truth-quality localization** for indoor navigation and performance evaluation;
- **External odometry input** that can be fused by the onboard estimator (PX4 EKF2) to support autonomous flight in the absence of GNSS.

From an integration standpoint, the important point is that the motion-capture pose is produced outside the autopilot and must be delivered to PX4 through a chain of software components. This chain must preserve the meaning of the measurement, including reference frames, orientation conventions, and timestamp semantics; otherwise, PX4 may fuse an apparently accurate measurement in an inconsistent way.

3.3 Data Distribution: From Vicon Workstation to ROS 2

To make motion-capture data available to external machines, the Vicon workstation exposes tracking information through a network interface. In the laboratory setup, pose data are distributed over the local network and made accessible to the UAV companion computer as ROS 2 topics through a VRPN-based pipeline. [9, 10]

In practice, the pose of the tracked UAV can be accessed on the companion computer through a topic such as:

`/vrpn_mocap/drone4/pose`

This provides a standard ROS 2 entry point for the rest of the software architecture: conversion nodes can subscribe to the pose topic, apply the required frame conventions, and publish PX4-compatible external odometry. ROS 2 tools such as `ros2 topic list`, `ros2 topic info`, and `ros2 topic hz` help verify that the tracking data are available, consistent, and updated at the expected rate.

A practical advantage of this approach is that it keeps the motion-capture system decoupled from the autopilot: Vicon remains responsible for tracking, while ROS 2 provides the integration layer used to validate, log, and convert the data before it reaches PX4.

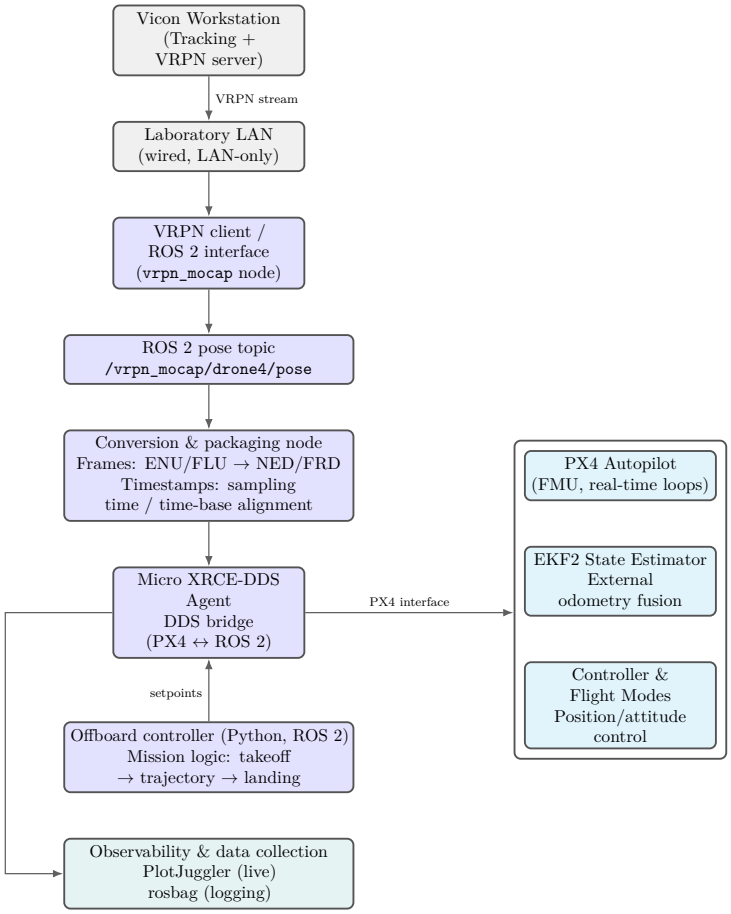

Figure 3.1: Motion-capture distribution and integration chain from Vicon/VRPN to ROS 2 and PX4

Figure 3.1 clarifies the role of the Vicon facility within the overall system. In particular, it highlights that Vicon is only the *source* of the pose measurement, while the key engineering work lies in transporting that measurement to the companion computer, converting it into PX4 conventions, and injecting it as external odometry with coherent timing. This separation is useful during debugging: each block can be validated independently before attempting end-to-end flight experiments.

3.4 Timing and Reference Consistency Requirements

Using motion capture as an estimator input requires more than geometric correctness. Two additional requirements are essential:

- **Reference consistency:** pose data are expressed in the reference frame used by the motion-capture system. Before the measurements can be fused by PX4, they must be represented according to PX4 conventions (frames and orientation conventions). This aspect is addressed in Chapter 6.
- **Timing consistency:** state-estimation fusion depends on timestamps and delay compensation. If the time base is inconsistent across machines, or if the end-to-end latency is unknown, the estimator may compensate delays incorrectly, potentially degrading performance.

In PX4 external-odometry messages, a distinction is made between the time at which a measurement is *sampled* and the time at which it is *received*. For reliable fusion, the sampling time should be consistent with the actual measurement acquisition time, and clocks across the involved machines should be aligned. Otherwise, the same physical trajectory can lead to different estimator behaviour across sessions, even if the motion-capture pose itself is accurate.

3.5 Clock Synchronization and Latency Characterization

The laboratory network is not always connected to the Internet; therefore, Internet-based NTP synchronization cannot be assumed. For this reason, a LAN-based time synchronization strategy was adopted using Chrony [11], with a reference machine acting as time server and the UAV companion computer configured as client. This provides a repeatable startup procedure and reduces ambiguity in timestamp interpretation during experiments.

Clock synchronization does not eliminate transport delay, but it provides a consistent temporal basis to interpret timestamps and to assess latency in a reproducible way.

In addition to synchronization, the VRPN-to-ROS 2 delay was experimentally characterized using rosbag recordings and offline analysis. The objective of this measurement is to quantify the timing stability of the motion-capture distribution pipeline and to verify that the network path is not a dominant limitation for estimator fusion. Table 3.1 summarizes the measured delay statistics.

Table 3.1: Measured VRPN→ROS 2 delay statistics for the motion-capture pose topic.

Metric	Value
Mean delay	0.40 ms
Standard deviation	0.24 ms
Minimum	0.07 ms
Maximum	5.6 ms

The results indicate a small average delay and low jitter in nominal conditions. Occasional higher-delay samples can occur, but the overall magnitude remains well below typical estimator delay-compensation ranges. These quantitative observations support subsequent configuration choices for external-odometry fusion in EKF2 and contribute to the reproducibility of indoor experiments.

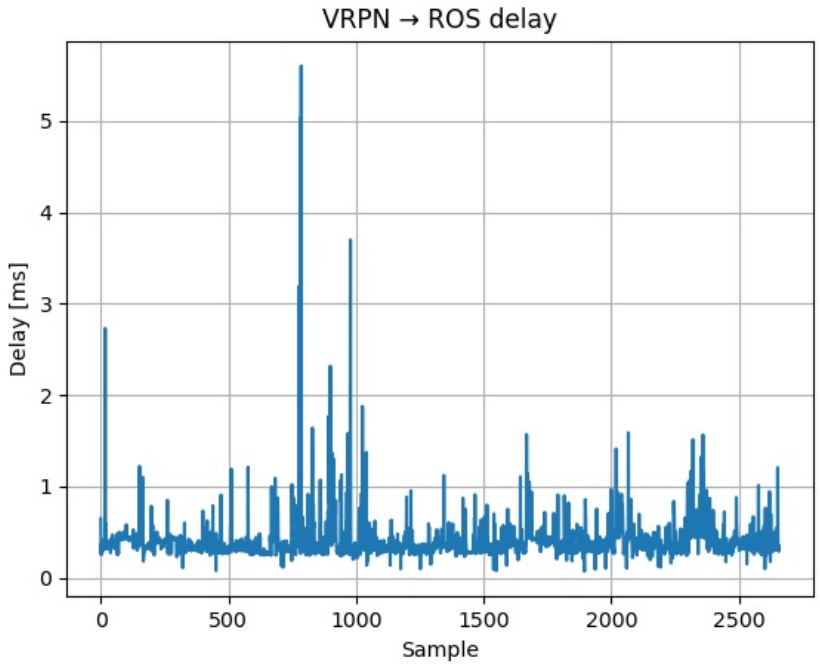

Figure 3.2: Measured VRPN-to-ROS 2 transport delay over the laboratory LAN

Each point represents the delay of a pose sample from the motion-capture stream as observed on the ROS 2 side. The delay remains sub-millisecond for the majority of samples, with occasional millisecond-scale outliers, supporting the assumption that network transport is not the dominant source of latency in the

integration pipeline. More importantly, the delay distribution is stable enough to be treated consistently during estimator configuration and across repeated experimental sessions.

3.6 Summary

This chapter introduced the indoor test facility and the real Vicon motion-capture infrastructure used in the project. The UAV pose is estimated in real time by the Vicon workstation and distributed over the LAN through a VRPN-to-ROS 2 pipeline, making it accessible as ROS 2 topics on the companion computer. A TikZ overview diagram was included to make the data path explicit and to clarify the separation between tracking, transport, conversion, and PX4 fusion. Since PX4 relies on timing and delay compensation for estimator fusion, clock synchronization and latency characterization are treated as key engineering requirements, enabling reproducible experiments and supporting the real-world integration described in the following chapters.

Chapter 4

Software Architecture

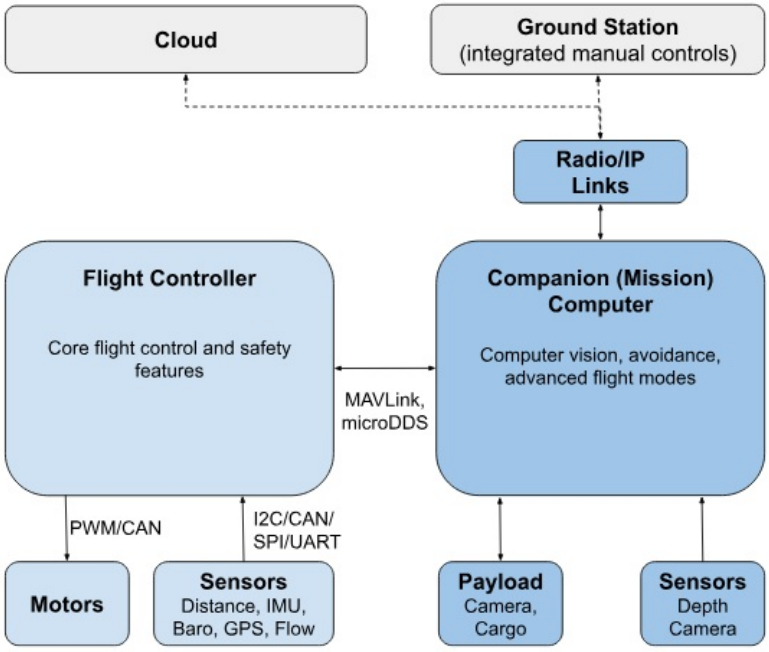

Figure 4.1: High-level UAV system architecture

4.1 Overview

The software architecture developed in this thesis connects three domains: (i) the PX4 flight stack running on the flight controller, (ii) the ROS 2 ecosystem running on the companion computer, and (iii) the motion-capture infrastructure providing external localization [12, 13]. The main objective is to deliver Vicon-based pose information to PX4 as *external odometry* in a format that is consistent in both reference frames and time, and to enable autonomous flight through an offboard control interface [14].

Figure 4.1 summarizes the high-level separation adopted in the platform: the flight controller remains responsible for safety-critical estimation and control loops, while the companion computer hosts the integration logic and higher-level autonomy. In practice, this split is what makes the development process manageable: the PX4 side stays relatively stable, whereas the companion computer can be iterated quickly to refine the motion-capture interface, frame conversions, logging, and mission logic. This separation also helps keep responsibilities clear: PX4 provides the robust real-time backbone, while ROS 2 acts as a flexible integration layer that can be modified without touching the low-level flight stack.

From an end-to-end perspective, the pipeline used in this work can be described as follows. The Vicon system estimates the 6-DoF pose of the tracked UAV and exposes it on the laboratory LAN through a VRPN stream. On the companion computer, the ROS 2 layer receives this stream, converts the pose to PX4 conventions, assigns a coherent sampling time, and publishes a PX4-compatible external odometry message. PX4 then fuses this information inside EKF2, while a ROS 2 offboard controller can command the vehicle for autonomous indoor missions. In parallel, the same ROS 2 environment provides a unified workflow for observability (logging and real-time visualization), which is essential during integration and experimental sessions [9, 14, 15].

A recurring engineering theme throughout this thesis is that a working indoor autonomy pipeline is not only a matter of "having the right topics". The system must be consistent at three levels simultaneously: (i) **semantics** (what a signal represents), (ii) **geometry** (frames, handedness, axis directions), and (iii) **time** (sampling times and delays). Most integration issues appear when one of these layers is overlooked. For example, a correct numeric pose becomes unusable if expressed in the wrong world frame, and a perfectly converted pose can still destabilize estimator fusion if its timestamps are not coherent with the onboard time base. For this reason, the architecture is structured so that each stage can be checked independently, from the VRPN pose at the input to the estimator outputs at the PX4 side.

4.2 PX4 Autopilot and State Estimation

PX4 Autopilot provides the flight modes, failsafe logic, and the real-time control loops required for stable flight. Indoor autonomy in this thesis relies on the EKF2 state estimator [16, 17], which fuses IMU measurements with additional sensor sources. Since GNSS is not available indoors, the key external source is motion-capture odometry.

From an integration perspective, two aspects are critical:

- **Frame conventions:** PX4 uses specific body and world frames (notably NED for world quantities and FRD for body quantities). External pose measurements must be converted accordingly before fusion.
- **Delay compensation:** EKF2 compensates sensor delays using timestamps and internally tracked delays. Therefore, the external odometry message must include a meaningful sampling time and a coherent time base.

To avoid treating estimator fusion as a black box, it is helpful to keep in mind that EKF2 continuously predicts the vehicle state at high rate using inertial measurements, and then applies corrections when external measurements are available. External odometry is therefore not "just another topic": it becomes part of the estimator feedback loop and influences the state that the controller uses. In indoor flight, where GNSS is absent, this role becomes even more central because motion capture effectively provides the primary global reference for position and heading [14, 16].

Figure 4.2 provides a compact view of PX4 internal data flow. In this thesis, the key path is the estimator loop: external odometry is injected as an aiding source and fused by EKF2 together with IMU measurements to produce a consistent vehicle state. This state is then consumed by the position and attitude controllers to generate setpoints and actuator commands. The diagram also highlights why timing and conventions matter: the estimator and control loops operate at high rate, so frame consistency and timestamp coherence are prerequisites for stable indoor autonomy.

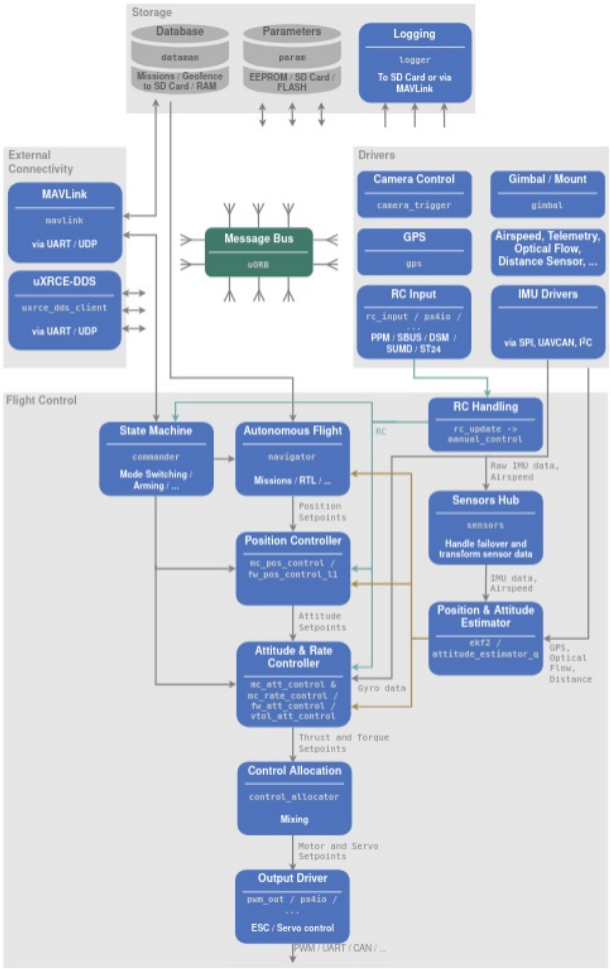

Figure 4.2: PX4 internal architecture overview

A practical implication is that the integration cannot be treated as a "black box". If yaw, position, or velocity disagree across the pipeline, the system may still produce numbers, but the controller will act on a state that does not match reality. Typical symptoms include slow drift, unexpected rotations, or unstable corrections when offboard setpoints are applied. For this reason, the architecture described in the next sections is organized around explicit conversion steps and around tools that allow checking estimator inputs and outputs during experiments.

4.3 ROS 2 Integration Layer

ROS 2 is used as the integration layer on the companion computer [13, 10]. Its role is to:

- subscribe to the motion-capture pose topic produced by the VRPN client interface;
- perform coordinate-frame conversions (world frame and body frame conventions);
- publish PX4-compatible external odometry messages for estimator fusion;
- run support tools for debugging and analysis (topic inspection, logging via rosbag, and offline processing).

In practice, ROS 2 is the "glue" that makes the system inspectable and easy to iterate. The same signals used for flight can be visualized live, recorded, and replayed offline [13, 10]. This reduces the dependence on one-off tests and makes integration issues easier to isolate: if a problem appears in the estimator output, the recorded data can be traced backwards to determine whether it originates from the motion-capture stream, from a frame conversion, or from a timestamp mismatch.

4.3.1 Design Choices for Real-Time Pose Streaming

The motion-capture stream is high rate and primarily used for state estimation, where a fresh sample is usually more valuable than an old one delivered reliably [10]. For this reason, the integration favors *timeliness* over *guaranteed delivery*. In practice, losing a small number of samples is acceptable, while building up latency is not. This consideration guided the choice of QoS profiles and the overall architecture: the pipeline is designed to remain responsive even during transient network load or during tool usage (e.g., live visualization).

Another practical design choice is to keep the ROS 2 integration chain simple and explicit. Instead of embedding all logic into a single complex node, the steps are conceptually separated: reception of the VRPN pose, frame/timestamp handling, and publication of external odometry. This makes it easier to validate each stage independently, and it allows incremental debugging (e.g., first verify the raw pose topic, then verify the converted pose, and only afterwards enable EKF2 fusion).

The exact QoS configuration used in the experiments is reported in Chapter 6. Here, it is sufficient to highlight the rationale: the configuration is tuned for high-rate streaming and low latency, which matches the needs of EKF2 aiding in indoor flight.

4.3.2 Common Failure Modes and Practical Checks

During integration, several issues can appear even when all nodes are running:

- **Frame inconsistencies:** the pose is numerically valid but expressed in the wrong frame (e.g., ENU vs NED, or wrong body axes), leading to yaw flips or mirrored trajectories.
- **Timestamp mismatch:** the sampling time is inconsistent with the onboard time base, causing EKF2 to fuse measurements with an incorrect delay.
- **Topic ambiguity:** multiple pose sources or multiple PX4 instances can lead to topic collisions if namespaces are not handled carefully.

These issues motivated a systematic debugging workflow based on observability (PlotJuggler, rosbag) and on step-by-step validation: before enabling offboard control, the pipeline is checked at the signal level to ensure that the estimator receives coherent external odometry.

4.4 PX4–ROS 2 Bridge with Micro XRCE-DDS

To exchange data between PX4 and ROS 2, this thesis adopts the Micro XRCE-DDS bridge [15]. Conceptually, the bridge exposes selected PX4 topics to the DDS/ROS 2 middleware and enables publishing ROS 2 messages that can be consumed by PX4. This is particularly useful for:

- accessing PX4 telemetry from ROS 2 for monitoring and logging;
- injecting external odometry data for EKF2 fusion;
- supporting offboard control by publishing setpoints and reading vehicle state.

A practical benefit of this approach is that the same ROS 2 tools used to validate the motion-capture pipeline can also be used to inspect PX4 messages (e.g., vehicle state, odometry, estimator outputs). This enables a unified workflow: one can observe the input pose stream, the converted external odometry, and the estimator response within a single environment.

4.4.1 Why XRCE-DDS Fits the Companion-Computer Workflow

From an engineering standpoint, Micro XRCE-DDS provides a clear separation between the real-time flight stack and the higher-level integration logic. PX4 continues to operate on the flight controller with its own internal messaging (uORB), while a selected set of topics becomes visible in ROS 2 through the XRCE-DDS bridge. This avoids ad-hoc telemetry parsing and keeps message semantics aligned with PX4 definitions [15].

In this thesis, XRCE-DDS was particularly convenient for two reasons. First, it simplifies observability: PX4 estimator and state topics can be plotted and logged with standard ROS 2 tools, which reduces friction during debugging. Second, it provides a consistent interface for offboard autonomy: the offboard controller can read the vehicle state and publish setpoints in a way that is naturally integrated into the ROS 2 ecosystem.

4.4.2 Interface Sanity Checks

Before running autonomous missions, the bridge is validated with simple checks that confirm correct end-to-end connectivity:

- **PX4 → ROS 2 visibility:** verify that key PX4 topics are visible on the ROS 2 side (telemetry presence and update rate).
- **ROS 2 → PX4 injection:** verify that external odometry messages can be published and received by PX4, and that EKF2 reacts consistently when fusion is enabled.
- **Offboard readiness:** verify that the controller can send setpoints without triggering unexpected failsafes, and that mode transitions are coherent (arming, offboard, landing).

These checks may look basic, but they save substantial time in real experiments: when something fails, it becomes possible to localize the problem to a specific layer (bridge, conversion node, timing, or estimator configuration) rather than debugging the full stack at once [15, 18].

4.5 Functional Blocks

The architecture can be described through the following functional blocks:

1. **Motion-capture acquisition:** Vicon workstation estimates the 6-DoF pose of the tracked UAV and distributes it over the laboratory network through a VRPN stream.

2. **ROS 2 pose interface:** on the companion computer, a VRPN client feeds ROS 2 and publishes the pose as a topic (e.g., `/vrpn_mocap/drone4/pose`).

3. **Conversion and packaging:** a ROS 2 node converts the pose to PX4 conventions (frames and orientation) and packages it into a PX4-compatible external odometry message.

4. **PX4 fusion:** EKF2 fuses the external odometry with inertial data to estimate the vehicle state.

5. **Offboard autonomy:** a Python controller runs in ROS 2 and commands the vehicle through an indoor mission profile (takeoff, trajectory tracking, safe landing).

This sequence summarizes the end-to-end logic of the system and the way in which external localization is transformed into estimator input and, ultimately, into closed-loop flight behavior [9, 14, 15].

4.6 Data Logging and Observability

Real-world integration is rarely a straight line: estimator fusion, frame alignment, timestamp handling, and communication issues can produce subtle failure modes. For this reason, the architecture explicitly includes an observability layer that makes the system *inspectable* during experiments.

Two complementary approaches are used:

- **Live visualization:** PlotJuggler [19] is used to monitor selected ROS 2 and PX4 streams in real time. This is especially helpful to compare motion-capture pose, injected external odometry, and estimator outputs side-by-side, and to detect issues such as yaw inconsistencies, sign errors, or delayed signals.

- **Offline analysis:** rosbag logging enables post-flight inspection of the same signals, allowing repeatable analysis across sessions and the extraction of quantitative metrics (e.g., delay statistics or tracking error).

This workflow reduces "trial-and-error" iterations: rather than modifying parameters blindly, issues can be isolated by checking each stage of the pipeline (source pose → converted odometry → estimator fusion → control response). A secondary advantage is that recorded datasets allow comparing different configurations on the same input data, which is often more informative than repeating flights with slightly different conditions.

4.7 Deployment and Reproducibility

A key design requirement for real-world experiments is reproducibility: the system should be deployable on the companion computer in a deterministic way, without manual intervention and without relying on external Internet connectivity.

For this reason, the integration layer was engineered to run headless at boot time through a small set of services with explicit dependencies. In practice, the following processes are treated as long-running components:

- the **Micro XRCE-DDS Agent**, required to bridge PX4 uORB topics to ROS 2;
- the **motion-capture interface** (VRPN client to ROS 2), which receives the tracked pose over the laboratory LAN;
- the **conversion and injection node**, which converts pose samples to PX4 conventions and publishes external odometry messages.

These components are executed as `systemd` services with a controlled start-up order (e.g., the conversion node starts only after the XRCE-DDS Agent is active) and automatic restart policies. This approach provides two practical advantages during experimental sessions: (i) the full software stack can be started reliably with a single power-up sequence, and (ii) failures become observable through persistent logs rather than transient terminal outputs.

Reproducibility also depends on operating in a *LAN-only* environment. During experiments, all required assets (packages, scripts, configuration files, and service definitions) are stored locally on the companion computer, and the workflow avoids assumptions about Internet access. This constraint influenced both deployment choices (local installation of dependencies) and validation practices (on-board logging with rosbag and offline post-processing).

4.7.1 Communication Endpoints for Reproducibility

In practice, reproducibility also benefits from making the communication endpoints explicit. The PX4 ecosystem exposes different channels with different purposes: MAVLink is used by the ground station for monitoring and mode management [7], whereas Micro XRCE-DDS is used to bridge selected PX4 topics to ROS 2 for integration and offboard control [15]. When multiple instances are involved (for example in multi-UAV SITL), keeping the endpoints separated is not only a convenience: it is the difference between a debuggable setup and a confusing one.

In this work, the same separation principle was applied explicitly in the multi-UAV SITL setup by running independent Micro XRCE-DDS agents on different UDP ports, so that each PX4 instance was mapped to a dedicated ROS 2 endpoint. The same design logic remains useful in real experiments, even in the single-UAV case, because it encourages clear endpoint allocation and easier troubleshooting. This port-level separation reduces ambiguity, avoids cross-talk, and makes troubleshooting faster: if a vehicle does not appear in ROS 2, the issue can be checked at the network level first (agent endpoint and connectivity) before investigating higher-level logic. Similarly, MAVLink traffic can be verified independently through the ground station, which is useful to confirm that the vehicle is alive even when the ROS 2 integration chain is temporarily disabled.

The benefit of this design choice is practical rather than theoretical: it supports a structured debugging flow. When something fails, it becomes possible to ask simple, local questions ("is the agent reachable on the expected port?"; "does QGroundControl see the vehicle?"; "is the pose topic present?") instead of re-running the full launch sequence blindly. In other words, clear endpoints turn a complex system into a set of smaller checks that can be performed quickly during a real experimental session.

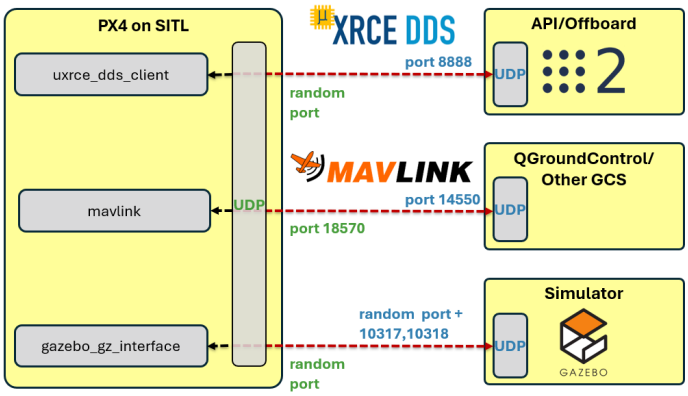

Figure 4.3: PX4 SITL communication overview

Figure 4.3 summarizes the separation between the main communication channels used during development. Even though the specific port numbers are a configuration detail, the underlying idea is general: each component has a clear role and a dedicated endpoint, which improves traceability and makes repeated experiment sessions easier to run and verify. In practice, this also supports safe iteration: XRCE-DDS can be restarted without affecting MAVLink monitoring, and ground-station connectivity can be checked independently of ROS 2 integration. This separation proved useful both in multi-instance simulation (where two agents were run in parallel) and in real experiments, where a deterministic network layout reduces integration time and helps avoid subtle configuration mistakes.

4.8 Summary

This chapter presented the software architecture adopted to enable indoor autonomous flight with Vicon-based localization. The system is built around PX4 for real-time estimation and control, ROS 2 as integration layer, and Micro XRCE-DDS as communication bridge. In addition, the system was engineered for reproducibility through headless deployment and an observability workflow based on live visualization and offline logging. The next chapters describe the simulation baseline used for early verification and the real-world integration steps required for consistent estimator fusion and autonomous flight execution.

Chapter 5

Simulation Baseline

5.1 Role of Simulation in the Development Workflow

In this thesis, simulation is used as a controlled baseline to validate the software environment before moving to the real indoor facility. Its role is not to reproduce every detail of the experimental setup, but to provide a repeatable and low-risk context in which the PX4–ROS 2 communication chain, the launch procedure, and the main integration tools can be verified in a systematic way.

This choice has an important methodological advantage. When the software stack is first exercised in simulation, many classes of errors can be identified early, including environment misconfiguration, DDS communication problems, missing ROS 2 messages, topic-discovery issues, or incorrect launch ordering. As a result, when real experiments are later performed in the laboratory, failures can be attributed more confidently to the real integration layer itself, such as motion-capture ingestion, timestamp interpretation, frame conversion, or estimator configuration, rather than to generic software instability.

Simulation is therefore treated as a development and validation tool. It provides a reproducible starting point, supports fast iteration cycles, and helps establish a disciplined workflow that can later be transferred to the real platform with fewer unknowns. In this sense, the simulation baseline is not an isolated preliminary exercise, but an integral part of the overall validation strategy adopted in the thesis.

5.2 PX4 SITL Environment with Gazebo

The baseline simulation environment is based on PX4 Software-In-The-Loop (SITL) coupled with Gazebo [20]. In this configuration, the PX4 flight stack runs as software on the development machine while interacting with a simulated vehicle model and a simulated world. This makes it possible to test estimator interfaces, control modes, communication endpoints, and development tooling without exposing physical hardware to unnecessary risk.

The simulated vehicle model used throughout this work is the `gz_x500` quadrotor. This model was selected because it provides a practical reference platform for validating the integration chain in an environment that is already compatible with the PX4 toolchain. The SITL setup also supports standard ground-station interaction, which is useful for parameter inspection, flight-mode monitoring, and general sanity checks during development.

From an engineering perspective, the main value of the SITL environment lies in repeatability. The same vehicle model, startup sequence, communication endpoints, and software dependencies can be exercised multiple times under controlled conditions. This enables a clean debugging workflow: topics can be inspected from a known initial state, DDS bridging can be verified systematically, and configuration errors can be isolated without the additional uncertainty introduced by physical hardware, wireless links, or sensing infrastructure. The launch discipline established in simulation is later reused as a practical reference during the transition to the real indoor platform.

5.3 Baseline Validation of the PX4–ROS 2 Interface

Before addressing multi-vehicle scenarios, the first objective in simulation is to validate the basic PX4–ROS 2 interface in a single-vehicle setup. This step acts as an infrastructure-level sanity check and confirms that the software environment is able to support the later stages of the project.

The main aspects verified in this phase are:

- correct operation of the Micro XRCE-DDS communication chain [15, 18];
- visibility of PX4 topics on the ROS 2 side for telemetry inspection and debugging;
- correct ROS 2 environment setup, including message generation and package discovery;

- ability to publish and inspect messages through the expected software interfaces.

Although these checks are conceptually simple, they are essential in practice. A large fraction of integration failures in robotics systems does not originate in advanced control logic, but rather in incomplete builds, mismatched message definitions, incorrect workspace sourcing, or communication layers that appear to start correctly but do not expose the expected topics. Verifying the interface early in simulation therefore reduces ambiguity later in the project.

The outcome of this phase is not a "final" validation of autonomy, but a stable operational baseline: a known-good development configuration in which PX4, ROS 2, and the bridging components can be started from a clean system state and inspected reliably. This baseline is then reused as the foundation for the more demanding multi-UAV scenario described in the following sections.

5.4 Multi-UAV Simulation with Fake VICON

After validating the single-vehicle baseline, the simulation workflow is extended to a multi-UAV scenario using a fake Vicon pipeline. This step is coherent with the general methodology of the thesis: before transferring the integration logic to the real indoor facility, it is useful to verify in simulation that the software architecture can be replicated consistently for more than one vehicle and that the data paths remain clearly separated.

The multi-UAV scenario serves two complementary purposes. First, it tests the scalability of the integration pipeline by replicating PX4 instances, ROS 2 interfaces, and control nodes while preserving separation between them. Second, it provides a controlled way to verify that external-pose streams, bridges, and offboard controllers do not interfere with each other when multiple vehicles coexist in the same environment.

This is particularly relevant because the complexity of the system does not grow only through the number of drones, but through the number of interacting software components. Once multiple vehicles are present, errors such as mixed namespaces, reused ports, or incorrect instance associations become much more likely and much harder to diagnose. For this reason, the multi-UAV simulation is not treated as an optional extension, but as a useful stress test for the overall architecture.

5.4.1 Two-Instance Setup and Namespace Separation

The multi-UAV simulation is implemented by running two PX4 SITL instances in Gazebo. Each instance is configured as an independent vehicle and is associated with:

- a unique vehicle identifier;
- a dedicated XRCE-DDS communication endpoint;
- separated ROS 2 communication paths and topic organization to avoid interference between data streams and control paths.

This separation is a fundamental requirement rather than a convenience. Without explicit instance isolation, telemetry topics, estimator outputs, or setpoint streams can become ambiguous, leading to misleading monitoring results and potentially incorrect control behaviour. In a single-vehicle setup, these issues may remain hidden; in a multi-vehicle setup, they become immediately problematic.

The simulated environment provides a safe and observable context in which instance separation can be checked carefully before dealing with the additional complexity of real localization, networking, and laboratory hardware. Each vehicle can be spawned in the same world, observed visually, and monitored through its own communication pipeline. This makes it possible to verify that the software architecture scales in a controlled way without changing its conceptual structure.

To make the setup directly observable, both vehicles are spawned in the same Gazebo world and monitored simultaneously. Figure 5.1 shows two `gz_x500` instances coexisting in the same simulated scene (entities `x500_0` and `x500_1`). This visual confirmation is useful because it establishes, at a glance, that the simulation is not simply running two abstract instances in the background, but two independent vehicles in a shared environment. This is the prerequisite for validating topic separation, bridge replication, and synchronized offboard activation.

In addition to simple coexistence in the same world, the two-instance setup is useful because it makes separation failures immediately observable. If namespaces, identifiers, or communication endpoints are not configured correctly, the resulting ambiguity typically appears very early in the workflow: ROS 2 topics may become difficult to interpret, monitoring tools may show inconsistent state information, and one controller may unintentionally interact with the wrong vehicle instance. In a real laboratory setup, these problems would be mixed with additional sources of complexity such as motion-capture transport, real clocks, and hardware initialization. In simulation, by contrast, the cause-effect chain is easier to isolate, which makes this stage particularly valuable for architectural validation.

A further advantage of the shared simulated environment is that it allows visual and software-level checks to be combined. Gazebo provides an immediate spatial

confirmation that two vehicles are actually active in the same scenario, while the corresponding PX4 and ROS 2 interfaces can be inspected in parallel to verify that each instance remains uniquely identifiable. This combined observability is important because multi-vehicle correctness cannot be established from a single layer alone: seeing two models in Gazebo is not sufficient if communication paths are mixed, and conversely, clean topic names are not sufficient if the simulated scene does not match the intended deployment. For this reason, the visual confirmation provided by the simulator is treated as the first practical verification step before proceeding to bridge replication and offboard activation.

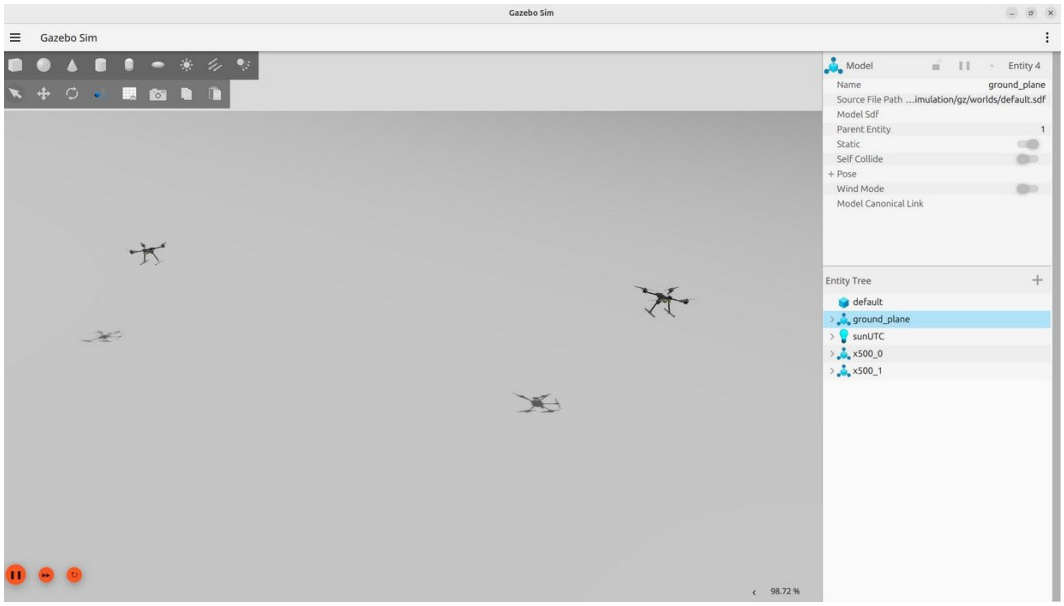

Figure 5.1: Two `gz_x500` PX4 SITL instances spawned in the same Gazebo

5.4.2 Incremental Launch Procedure and Bridge Replication

A key practical lesson from the multi-UAV simulation is that the setup should not be treated as a monolithic launch. When multiple vehicles and multiple communication paths are involved, starting all components at once makes fault isolation unnecessarily difficult. For this reason, the scenario was built incrementally, enabling each layer only after the previous one had been verified.

The launch sequence adopted during development is summarized below:

1. **Start two Micro XRCE-DDS agents:** two agents are launched on different ports (8888 and 8889), so that each PX4 instance is associated with its own

communication endpoint. This avoids port conflicts and keeps the DDS bridge structure explicit.

2. **Start the two PX4 SITL instances:** each PX4 instance is launched with its own instance index and unique identifier. At this stage, the objective is simply to verify that both simulated vehicles start correctly and remain distinguishable.

3. **Perform a ground-station sanity check:** QGroundControl is connected to the simulation to confirm that two distinct vehicles are detected and that their basic telemetry and flight-mode information can be inspected independently.

4. **Start the fake Vicon pose generation:** independent pose streams are generated for the two vehicles. The purpose is not to reproduce the full physics of motion capture, but to validate interface-level correctness: topic existence, stable publication, and correct vehicle association.

5. **Start the ROS 2 integration nodes:** for each vehicle, the same conceptual chain used later in the real setup is reproduced, including subscription to the pose stream, frame conversion, packaging into PX4-compatible messages, and publication toward the intended PX4 instance.

6. **Start two offboard controllers in standby:** each vehicle is associated with an offboard controller that remains idle until explicitly triggered. This avoids accidental command publication during the setup phase.

7. **Trigger a synchronized mission start:** a dedicated mission node is used to activate both offboard controllers in a coordinated way, providing a clean baseline for simultaneous operation.

This incremental procedure proved useful not only for reproducibility, but also for debugging efficiency. Configuration mistakes such as wrong ports, swapped identifiers, or namespace collisions could be detected early, before propagating into higher-level behaviours. More importantly, the same development tools used in the single-vehicle case—QGroundControl, ROS 2 topic inspection, and logging—remained effective in the multi-UAV scenario because the architecture preserved clear separation between instances.

From a methodological point of view, this launch discipline is one of the most valuable outputs of the simulation phase. It transforms the multi-vehicle setup from a fragile development experiment into a repeatable workflow that can be inspected step by step and re-executed after changes to the software stack.

5.4.3 Ground-Station Observability in a Multi-Vehicle Scenario

Beyond ROS 2-side introspection, it is useful to confirm that the two simulated vehicles remain distinguishable from the viewpoint of a standard ground-station tool. This provides an additional verification layer at the telemetry level: each PX4 instance should appear as a separate vehicle and should expose its own state, mode, and status information.

Figure 5.2 shows QGroundControl [7] connected to the simulation while detecting multiple vehicles. This view is important because it confirms that the instance separation is preserved not only inside ROS 2, but also at the level of system identification and general autopilot observability. In practice, this reduces the risk that a configuration mistake remains hidden behind apparently valid ROS 2 topics while the autopilot instances are not actually separated as intended.

Even though the real experiments discussed later in the thesis focus on a single UAV, validating multi-vehicle observability in simulation has two benefits. First, it strengthens confidence in the communication layout used during development. Second, it prepares the software stack for future extensions in which multiple vehicles may need to coexist while remaining clearly identifiable to both ROS 2 tools and ground-station software.

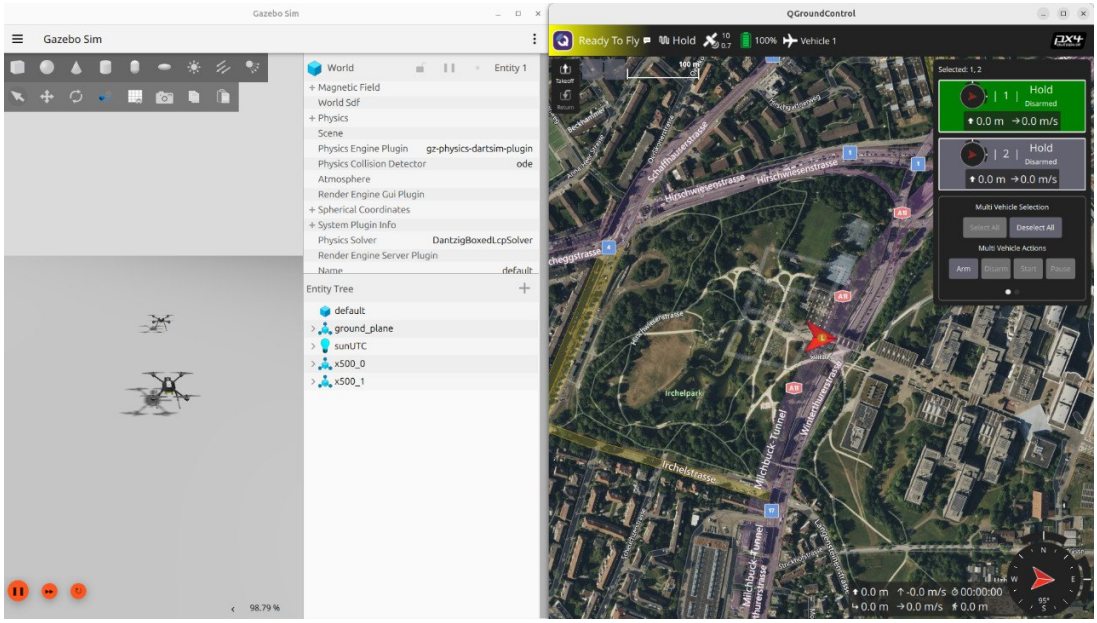

Figure 5.2: QGroundControl monitoring multiple vehicles in the multi-UAV SITL scenario

5.4.4 Fake VICON Pose Streams and Interface Validation

The fake Vicon pipeline is introduced to reproduce, at the software-interface level, the role that the real motion-capture system will later play in the indoor facility. In simulation, it generates pose information for each vehicle and publishes it into ROS 2 as two distinct streams associated with the corresponding instances.

The objective is not to emulate the internal operation of the Vicon system or the exact properties of optical motion capture. Instead, the fake pipeline is used to validate the assumptions that matter for integration:

- the existence of two stable and independent pose streams;
- unambiguous mapping between each pose stream and its intended PX4 instance;
- correct replication of the conversion and publishing chain for multiple vehicles;
- possibility to inspect relevant state variables for each UAV without mixing data sources.

In this sense, the fake Vicon layer acts as a functional placeholder for the real motion-capture interface. It allows the software chain to be exercised under multi-instance conditions while keeping the environment simple and controlled. This is particularly useful because it isolates the problems of naming, routing, and controller association from the additional complexity that will later arise with real clocks, real network transport, and real external localization.

In the real facility, the motion-capture chain introduces additional complexity, including network transport, tracking-side processing, timestamp interpretation, and possible frame inconsistencies. If all these elements are introduced together with multi-vehicle replication, it becomes harder to determine whether a problem is caused by the localization infrastructure or by the software architecture itself. The fake Vicon approach removes this ambiguity during the simulation phase.

At this stage, the key question is not whether the simulated pose is physically identical to a real Vicon measurement, but whether the architecture can accept multiple external pose streams, keep them separated, and route them consistently to the intended PX4 instances. In this sense, the fake Vicon layer is used to validate interface discipline and pipeline replication before transitioning to the laboratory environment.

The use of fake pose streams also supports a more rigorous debugging process. If an issue appears in the multi-UAV simulation, it can usually be traced back to the structure of the software pipeline itself rather than to the laboratory infrastructure. This supports the conclusion that, once transferred to the real facility, the remaining challenges are primarily related to physical integration.

5.4.5 Validation Criteria and Observed Outcomes

The multi-UAV simulation is evaluated using practical criteria focused on integration correctness rather than on dynamic flight-performance metrics. The goal is not to demonstrate complex coordinated control, but to confirm that the software architecture remains coherent when duplicated across two vehicles.

The main validation criteria are the following:

- **Topic separation:** each UAV must expose its own ROS 2 topic namespace, with no collisions between telemetry, pose streams, or command paths.
- **Bridge visibility:** PX4 topics for both instances must be visible and inspectable independently from the ROS 2 side.
- **Data-flow coherence:** each fake-Vicon pose stream must be consumed by the intended conversion and publishing chain, without cross-association between vehicles.
- **Offboard activation consistency:** both vehicles must be able to transition from standby to offboard operation when triggered by the mission node, without partial starts or race conditions.

The observed outcomes were consistent with these criteria. During repeated launches, the two vehicles remained distinguishable both in Gazebo and in QGroundControl, and no topic collisions were observed in the ROS 2 environment. The fake-Vicon pose streams remained separable and could be associated with the intended instances throughout the tested workflow. In addition, the offboard controllers remained in standby until the mission trigger was issued, after which both vehicles could be activated from a clean initial state and start the mission together.

These results do not constitute a final validation of multi-robot autonomy, nor do they replace real experiments with the laboratory motion-capture system. However, they provide a strong baseline for the integration methodology adopted in the thesis. They show that the software architecture can be replicated consistently across multiple vehicles and that the supporting tools remain usable even when the scenario becomes more complex. In the next chapters, the same integration principles are transferred to the real indoor facility, where the pose stream is provided by the actual Vicon system and the methodology is validated experimentally on a single UAV.

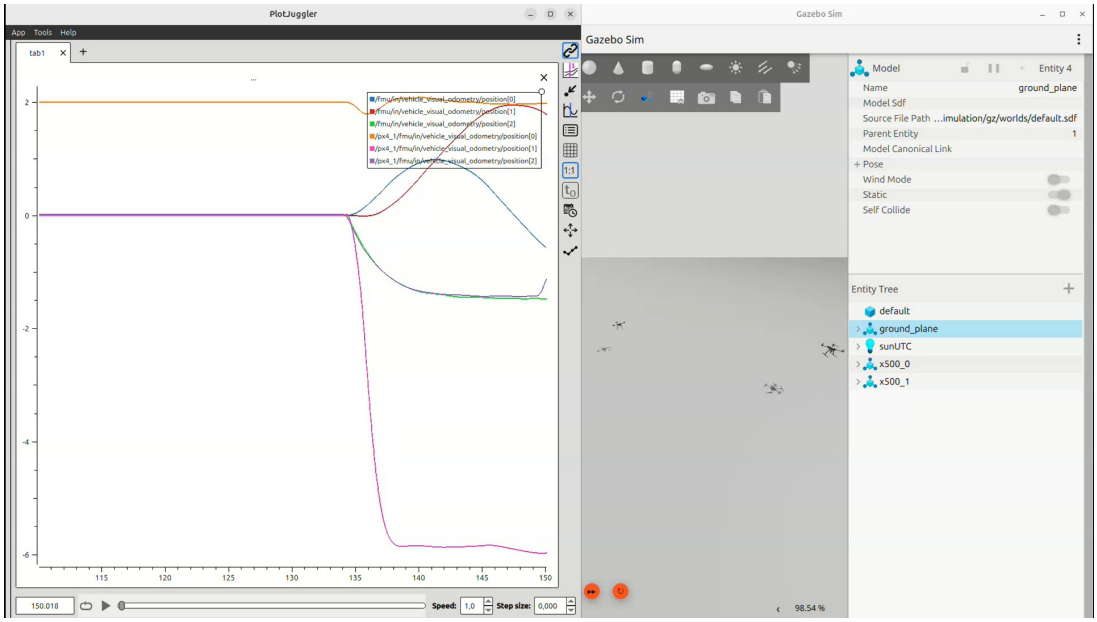

Figure 5.3: Multi-UAV SITL scenario with fake VICON-based pose streaming

Figure 5.3 provides a representative snapshot of the multi-UAV simulation during execution. The Gazebo view confirms the simultaneous presence of the two simulated vehicles in the same environment, while the PlotJuggler panel shows the corresponding position signals associated with the fake VICON-based pose streams. In this sense, the figure is useful because it shows not only that the simulator was active, but also that the vehicles were actually executing motion while remaining observable through the expected software pipeline.

A relevant aspect of this visualization is that it gives direct evidence that the fake VICON layer was not limited to publishing static topics. The position signals evolve over time and show distinct behaviors for the two UAV instances, consistently with the fact that the vehicles were executing a simulated trajectory in Gazebo. This is important because, before moving to the real facility, the software architecture must be verified under conditions in which the external pose streams are generated, propagated, and visualized coherently while the vehicles are effectively moving.

From a methodological point of view, this combined visualization was particularly useful during development. It allowed the simulation scene and the signal-level behavior to be checked at the same time, making it easier to confirm that the two vehicles remained distinguishable, that the corresponding streams did not interfere with each other, and that the fake VICON pipeline was effectively supporting the intended multi-instance validation workflow.

5.5 Limitations of the Simulation Baseline

Although the SITL environment is highly valuable for validating the software stack and the development workflow, it does not reproduce all aspects of the real indoor facility.

The main limitations are:

- real motion-capture timing, VRPN transport behaviour, and LAN-specific effects are not represented in full;
- frame-alignment and yaw-consistency issues related to external localization may appear less critical in simulation than in real experiments;
- hardware-specific factors such as power sequencing, radio links, companion-computer startup conditions, and physical disturbances are inherently absent;
- operational safety procedures, which are essential in real flight tests, are outside the scope of the simulated environment.

For these reasons, simulation is not treated in this thesis as the main validation environment, but as a controlled baseline and a development accelerator. It is useful to verify that the architecture is internally consistent and operational before moving to the laboratory, but it cannot replace the real integration work required for motion-capture-based indoor flight.

5.6 Summary

This chapter presented the simulation baseline used to support the development workflow of the thesis. PX4 SITL with Gazebo provides a repeatable environment for validating the software setup, the PX4–ROS 2 interface, and the associated development toolchain. Building on this baseline, a multi-UAV simulation with a fake Vicon pipeline was used to verify instance separation, bridge replication, observability, and synchronized offboard activation under controlled conditions.

The main value of this simulation phase is methodological: it establishes a stable reference workflow and reduces ambiguity before moving to the real indoor platform. The following chapter transfers the same integration principles to the laboratory environment, where the pose source is provided by the real Vicon system and the pipeline is validated experimentally on the actual UAV.

Chapter 6

Real-World Integration and Validation

6.1 Purpose and Integration Strategy

After the simulation baseline, the focus of this chapter is the real-world integration of the complete pipeline: motion-capture pose acquisition, ROS 2 transport, conversion into PX4 conventions, and injection into PX4 for EKF2 fusion and offboard operation.

The integration was approached as a sequence of verifiable stages. Rather than treating the system as a black box, each stage was validated independently before moving to the next one. This point may sound obvious, but in practice it makes the difference between a setup that can be debugged quickly and a setup that "works once" and then wastes an afternoon when something changes (a reboot, a network hiccup, a different rigid body name, or a timing mismatch). The goal was to build a workflow that could be repeated across sessions with consistent outcomes.

The staged methodology adopted in this thesis is summarized below:

1. confirm that motion-capture data is available, stable, and correctly associated with the intended rigid body on the LAN;
2. confirm correct ROS 2 reception (topic presence, rate, and basic sanity checks);
3. validate frame conventions and orientation consistency (especially yaw/heading);
4. validate time base and delay handling (clock synchronization and timestamps);
5. validate bridge connectivity and message semantics toward PX4;

6. perform an end-to-end autonomous flight demonstration (reported in Chapter 7).

This structure reduces trial-and-error iterations and makes debugging repeatable across experimental sessions. When an issue appears, it can be traced back to the last validated stage instead of re-checking the entire system from scratch.

6.2 Experimental Setup

6.2.1 System Components

The real-world setup matches the architecture described in Chapter 4. The main components are:

- **Motion-capture system:** Vicon tracking infrastructure provides 6-DoF pose estimates of the UAV in the laboratory reference frame and publishes them over the local network (VRPN stream). [8]
- **Companion computer:** Ubuntu + ROS 2 environment used to receive pose data, perform conversions, publish external odometry to PX4, run offboard control, and log data. [13, 10]
- **Flight controller (PX4):** runs EKF2 for state estimation and the flight control loops. External odometry is fused as an aiding source for indoor flight. [17, 14]

A practical detail that matters during integration is that these components evolve at different speeds: PX4 runs high-rate loops and reacts immediately to inconsistent measurements, while the companion computer and the network introduce non-deterministic effects such as transport jitter, queueing, and occasional delays. For this reason, the validation strategy prioritizes observability and incremental checks over "one-shot" attempts.

6.2.2 Network Assumptions and LAN-Only Operation

A practical constraint of the laboratory environment is that the experimental workflow should not rely on Internet access. Therefore, the integration was designed around LAN-only operation:

- all relevant ROS 2 packages and scripts are available locally on the companion computer;
- clock synchronization and debugging tools are operated within the local network;

- logging is performed locally through rosbag for offline analysis.

This constraint impacts both deployment (e.g., installing dependencies offline) and validation (e.g., using local logs instead of cloud tooling). It also influences how "reproducibility" is interpreted: the system must be restartable from a clean state without external dependencies, and all evidence needed for analysis must be collectable locally.

6.3 Motion-Capture Pose Acquisition in ROS 2

6.3.1 VRPN Stream and ROS 2 Pose Topic

The Vicon system provides the rigid body pose through VRPN [9, 10]. On the companion computer, a VRPN client interface publishes the pose into ROS 2 as a standard pose message, typically on a topic of the form:

`/vrpn_mocap/drone4/pose`

At this stage the key goal is to verify that the pose stream is present, stable, and consistent with the physical motion of the UAV. Importantly, this check is performed *before* any conversion logic: keeping the first step simple helps distinguish tracking problems (wrong rigid body selection, tracking loss, occlusions) from issues introduced later by the integration layer.

6.3.2 Basic Sanity Checks: Presence, Rate, and Coherence

The first validation step is intentionally simple: before any conversion or fusion, the raw pose stream must satisfy a few observable properties:

- **Topic availability:** the pose topic is visible and has an active publisher.
- **Update rate:** the stream rate is compatible with real-time estimation (typically tens to hundreds of Hz, depending on the Vicon configuration).
- **Coherence:** position coordinates change coherently with motion, without persistent discontinuities or sign inversions.

These checks help distinguish between motion-capture issues (tracking loss, wrong rigid body selection) and integration issues downstream. They also provide a baseline expectation: if the raw stream already contains discontinuities, later stages cannot "fix" it by conversion or filtering.

6.4 Frame Conventions and Orientation Handling

6.4.1 Frame Mapping from Motion Capture to PX4

A critical integration step is aligning coordinate conventions. Motion-capture systems and robotics middleware commonly provide poses in an ENU/FLU-like convention, while PX4 expects world quantities in NED and body quantities in FRD [14, 17]. Therefore, the conversion node performs:

- **World-frame mapping:** ENU $\rightarrow$ NED, including axis swaps and sign changes.
- **Body-frame convention:** body axes are mapped from FLU $\rightarrow$ FRD in the orientation representation (and, when used, velocity is expressed accordingly).

Even small mistakes here can lead to stable-looking data that is nevertheless inconsistent for fusion or control.

From an implementation point of view, this conversion is not a purely formal step. The motion-capture system, the ROS 2 middleware, and PX4 do not simply use different labels for the same quantities; they encode different geometric assumptions about how world and body axes are defined. As a consequence, the conversion layer must preserve the physical meaning of the measurement while expressing it in the conventions required by the autopilot. This is why the frame-mapping stage is treated explicitly in the integration pipeline rather than being embedded implicitly inside higher-level control logic or estimator tuning.

A correct transformation is therefore a prerequisite for meaningful estimator fusion, not a cosmetic post-processing step. It also has direct consequences for experimental repeatability: when the same physical motion is represented consistently across all components, observed estimator responses can be interpreted with confidence and compared across sessions. Conversely, if the frame semantics are only partially correct, the system may still appear to work during simple checks while becoming misleading as soon as more dynamic motion or closed-loop offboard control is introduced.

In practice, it is easy to underestimate this point because a wrong conversion may still yield "reasonable" numerical values. The real issue is semantic: if the estimator believes "up" is "down" or if the yaw axis is inverted, EKF2 is effectively being corrected in the wrong direction. The result can range from subtle drift to immediate instability once offboard setpoints are applied.

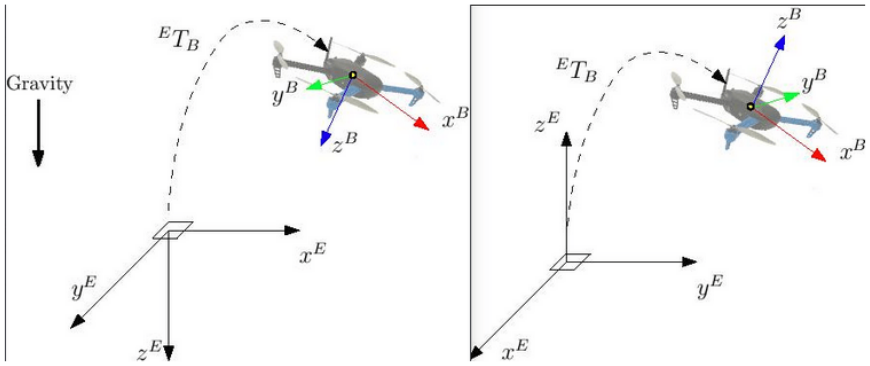

Figure 6.1: ENU/FLU versus NED/FRD frame conventions

The figure highlights the two conventions that must be reconciled in this thesis. On the motion-capture and ROS 2 side, pose data are naturally expressed using an ENU world frame and a FLU body frame, which is common in robotics: the world z axis points upward, and the body x axis points forward with y to the left. PX4, on the other hand, uses NED for world quantities and FRD for body quantities: the world z axis points downward, and the body y axis points to the right. These are not cosmetic choices; they imply sign changes and axis permutations that directly affect both position and attitude. [14, 17]

In practice, the conversion must be applied consistently to (i) the translation vector and (ii) the orientation. For the orientation, it is not enough to "rename axes": the quaternion (or rotation matrix) must be transformed so that the body axes are mapped correctly, otherwise yaw and roll/pitch may appear swapped or mirrored. If the conversion is incorrect, typical symptoms include mirrored trajectories in the horizontal plane, sudden yaw jumps, or a persistent yaw offset between the motion-capture reference and the estimator output.

For these reasons, frame handling in this thesis is treated as a first-class integration step rather than an implementation detail. The conversion layer is validated using sanity checks that compare signals across the pipeline (Vicon pose, injected external odometry, and EKF2 estimates), ensuring that the same physical motion corresponds to the same sign and direction in all components before enabling offboard control. This is also why yaw is used as a practical diagnostic quantity: it provides an immediate and intuitive way to spot frame mistakes through simple physical tests.

6.4.2 Yaw Consistency as a Practical Diagnostic

In indoor flight, yaw (heading) is often the first component that reveals frame inconsistencies. During integration, a recurring symptom of a wrong convention is that position tracking appears reasonable while yaw behaves unexpectedly (e.g., sign inversion, discontinuities around $\pm\pi$, or a constant offset).

For this reason, yaw was used as a practical diagnostic signal:

- compare yaw derived from motion capture against the yaw implied by the injected external odometry;
- compare both against PX4 estimator outputs during stationary tests and slow rotations;
- verify that the observed behavior matches the expected physical rotation direction.

This approach makes debugging concrete: instead of reasoning only on matrices, the system is validated against a simple physical action (rotate the vehicle) and the expected sign/direction of yaw. When the yaw chain is correct, the remaining integration issues are typically easier to localize, because the main sign and orientation conventions have already been validated.

6.5 Time Base and Delay Handling

6.5.1 Why Clock Synchronization Matters

EKF2 delay compensation relies on timestamps. If the companion computer and the PX4 time bases are inconsistent, external odometry may be fused with an effective delay that changes over time, which can degrade estimation and produce non-repeatable behavior.

A robust solution is to enforce a consistent time base within the local network. In this work, a LAN-based synchronization scheme was adopted using Chrony, so that the machine handling the external-odometry pipeline shared a consistent time base with the local reference node [11]. This enables consistent interpretation of acquisition timestamps and repeatable delay analysis. Beyond fusion quality, clock consistency is also important for debugging: if logs from different machines share a coherent time base, it becomes possible to align events (pose reception, odometry publication, estimator response) without guessing offsets manually.

Figure 6.2 shows the time-synchronization status obtained with Chrony during the experiments. The UAV-side integration pipeline runs with a LAN-consistent time base relative to the local reference machine, resulting in stable and repeatable

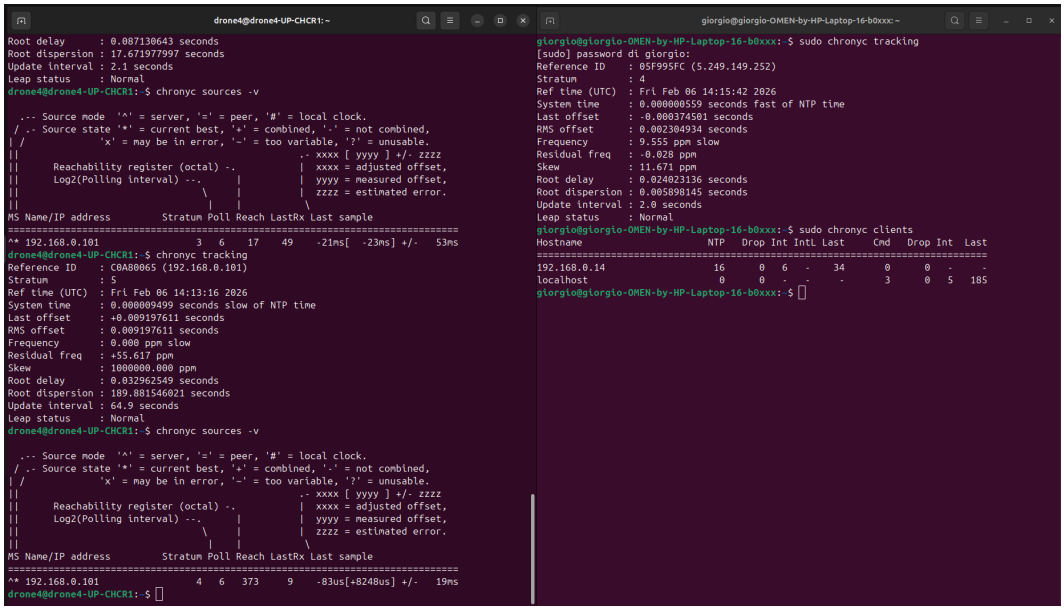

Figure 6.2: Chrony time synchronization status

timestamp interpretation across devices. This detail is more important than it may appear at first: external odometry messages are timestamped on the companion side and then fused by EKF2 on the flight controller, so any clock mismatch directly translates into an incorrect effective delay. With synchronization in place, timestamp semantics become consistent across sessions and the delay analysis described next can be interpreted reliably. In addition, a coherent time base simplifies troubleshooting because logs recorded on different machines can be aligned without manual offset guessing, making it easier to correlate cause (pose arrival, publication, injection) and effect (estimator response).

6.5.2 Delay Characterization Through Logging

Clock synchronization alone does not guarantee that the full pipeline delay is negligible. To understand the performance of the motion-capture interface, the end-to-end latency from pose publication to ROS 2 reception was characterized through logging and offline analysis.

The analysis provides two benefits:

- it verifies that delay and jitter remain small in the local network, supporting the assumption that the LAN is not the bottleneck;
- it supports informed choices for estimator parameters and message timestamp semantics.

In other words, the goal is not to chase "zero delay" (which is unrealistic), but to measure the delay distribution and ensure it is stable enough to be treated consistently by the estimator.

6.6 PX4 Bridge Validation and Message Semantics

6.6.1 Bridge Connectivity and Topic-Level Verification

With the motion-capture stream validated and conversions in place, the next step is verifying the PX4–ROS 2 bridge. The key requirement is not only that topics are visible, but that they are actively published and consumed with the expected rate and QoS. [15, 18]

A practical lesson from integration is that *seeing a topic in a list is not sufficient.* During debugging, mismatches between topic names, publisher presence, and QoS compatibility can lead to situations where data appears to exist but is not effectively delivered to PX4. For this reason, connectivity checks were performed at two levels: first confirm topic presence and update rate on the ROS 2 side, then confirm, through estimator-side observability, that the injected data is effectively entering the intended fusion pipeline when external aiding is enabled.

6.6.2 External Odometry Injection: What Must Be Coherent

For EKF2 fusion, the injected external odometry must be coherent in:

- **reference frames:** world and body conventions must match PX4 expectations;
- **orientation representation:** quaternion ordering and handedness must be correct;
- **timestamp semantics:** the message should reflect when the sample was acquired, not only when it was forwarded;
- **quality indicators:** fields such as covariance or quality indicators should be handled consistently with the intended confidence in the measurement.

If any of these elements are inconsistent, EKF2 may either reject the aiding source or fuse it in a way that produces subtle estimator drift or oscillatory behavior. This is one of the reasons why the integration workflow is staged: it is far easier to catch

an incorrect frame conversion at the signal level than to infer it from unstable flight behavior. [14, 17]

6.7 End-to-End Readiness Checklist

Before performing autonomous flight, an operational checklist was adopted to ensure that the pipeline behaves consistently across sessions:

- motion-capture pose topic is published and stable (rate and coherence);
- conversion node publishes external odometry with expected frame conventions;
- bridge is active and PX4 topics are visible and updating;
- clocks are synchronized within the LAN and timestamps are coherent;
- live visualization confirms expected behavior for a simple physical test (e.g., small translations and yaw rotations).

This checklist was especially useful to avoid spending flight time diagnosing issues that could have been detected on the ground. More importantly, it makes the integration repeatable: the same checks can be performed at the beginning of every session, so that a "working" day does not depend on accidental state left by previous runs.

6.8 Summary

This chapter described the real-world integration methodology and the validation steps adopted to build a reliable motion-capture-to-PX4 pipeline. The proposed workflow emphasized staged verification, frame consistency, time-base coherence, and bridge-level observability, so that each part of the system could be checked before attempting autonomous flight. In this sense, the main outcome of the chapter is not yet mission-performance assessment, but the demonstration that the localization, timing, and estimator interfaces are sufficiently consistent to support repeatable real-world testing. The next chapter presents the experimental validation on a single UAV, using the configured pipeline for indoor autonomous flight under offboard control.

Chapter 7

Experimental Demonstration

7.1 Objective and Scope of the Experimental Validation

This chapter presents the experimental demonstration of the complete indoor autonomy pipeline on a single UAV. The objective is not to provide an exhaustive benchmark of estimator tuning or trajectory-tracking performance, but to show that the system described in Chapters 4 and 6 can be executed end-to-end in a repeatable and technically coherent way:

- motion-capture pose is acquired and validated in ROS 2;
- pose is converted into PX4 conventions and injected as external odometry;
- EKF2 fuses the external aiding source in a stable manner;
- an offboard controller executes a simple indoor mission profile.

The experiments are therefore designed as a representative proof of functionality, with emphasis on integration correctness, estimator stability, and operational repeatability. In this context, the main question is not whether the system achieves optimal performance in every metric, but whether the complete localization-to-control chain behaves consistently enough to support real indoor autonomous flight.

7.2 Experimental Conditions and Mission Profile

7.2.1 Test Environment

All flights were conducted indoors under motion-capture coverage in the laboratory flight area. The Vicon system provided the 6-DoF pose of the tracked rigid body, while the companion computer executed the ROS 2 integration stack described in the previous chapter [8]. This setup enabled the vehicle to operate in a GNSS-denied environment while relying on external localization as the main aiding source for onboard estimation.

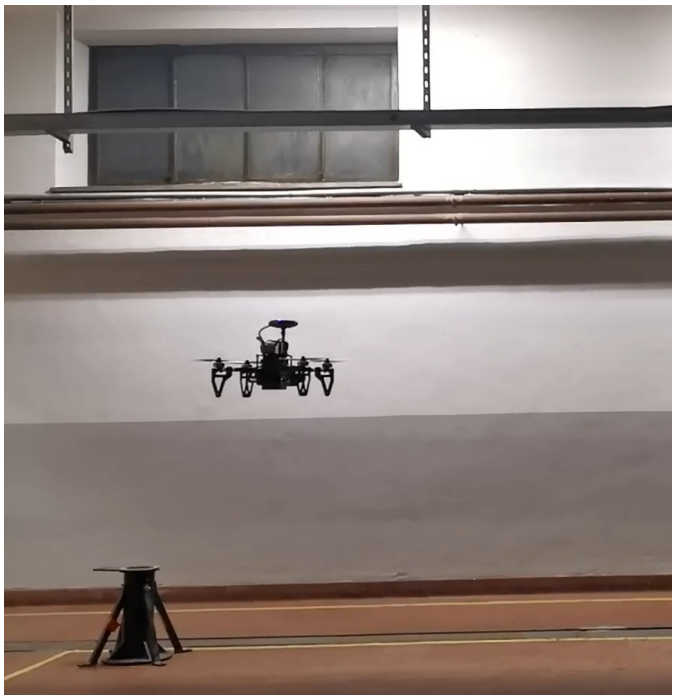

Figure 7.1: Indoor flight test under motion-capture coverage

Figure 7.1 illustrates the real experimental context of the thesis. Although the system architecture may appear modular when described block by block, the actual validation required all components to operate simultaneously and consistently: motion capture, ROS 2 transport, conversion logic, PX4 estimator fusion, and offboard control. For this reason, the experimental phase was treated not only as a final demonstration, but also as a confirmation that the staged integration strategy developed in the previous chapters was sufficient to support real flight.

7.2.2 Controller and Mission Definition

The offboard controller runs in ROS 2 (Python) and commands the vehicle through a minimal mission profile selected to exercise the complete pipeline while keeping the experiment interpretable and repeatable:

1. arming and takeoff to a safe indoor altitude;

2. short trajectory execution at moderate speed;

3. controlled landing and disarming.

This mission profile stresses both the estimator and the control interface. From the estimator side, it requires continuous and coherent external aiding during different phases of flight. From the control side, it verifies that setpoints can be accepted and executed without unstable transitions between modes. The mission is intentionally simple: its purpose is to validate the integrated pipeline under realistic operating conditions while keeping debugging manageable and repeated trials feasible.

7.3 Signals Logged and Evaluation Criteria

7.3.1 Logged Data

To support post-flight validation, the following data streams were logged:

- motion-capture pose in ROS 2, representing the raw localization input;

- converted external odometry published toward PX4;

- PX4 estimator outputs (e.g., vehicle odometry / estimator state);

- offboard setpoints and basic vehicle state information, such as mode and arming status.

Logging was performed through rosbag and complemented with live monitoring in PlotJuggler [19] during the experiments. This combination was useful in practice because live visualization made it possible to detect obvious anomalies before takeoff, while rosbag provided the temporal detail needed for post-flight comparison across the different stages of the pipeline.

7.3.2 Evaluation Criteria

The demonstration is evaluated using practical criteria aligned with the objectives of the thesis:

- **Consistency:** injected odometry must preserve the expected sign conventions and orientation behavior across the pipeline.
- **Estimator stability:** EKF2 must fuse the external aiding source without erratic corrections, repeated resets, or visually evident discontinuities.
- **Mission feasibility:** the UAV must complete takeoff, trajectory execution, and landing under offboard control in a repeatable way.

Whenever possible, these criteria are supported by plots comparing motion-capture pose, injected external odometry, and estimator outputs. The aim is not to claim perfect overlap between all signals, but to demonstrate that the system operates in a coherent and stable regime.

7.4 End-to-End Demonstration Results

7.4.1 Pose Consistency Across the Pipeline

A first validation step is checking that the same physical motion produces coherent signals across the localization and estimation pipeline. In practice, this means that:

- position trends remain consistent between motion capture and estimator outputs, up to bounded deviations associated with estimation dynamics;
- yaw evolves with the expected sign and without abnormal discontinuities during slow rotations and translational maneuvers;
- injected external odometry does not introduce abrupt jumps that are not already present in the source pose stream.

Figure 7.2 provides a representative signal-level comparison across the pipeline during a real flight. The important observation is not exact numerical coincidence at every sample, but coherent behavior: the main variations in position appear in the expected directions and with compatible timing across the compared signals. This supports the conclusion that the external pose information is being transformed and delivered consistently enough to preserve its geometric meaning when used by the onboard estimator.

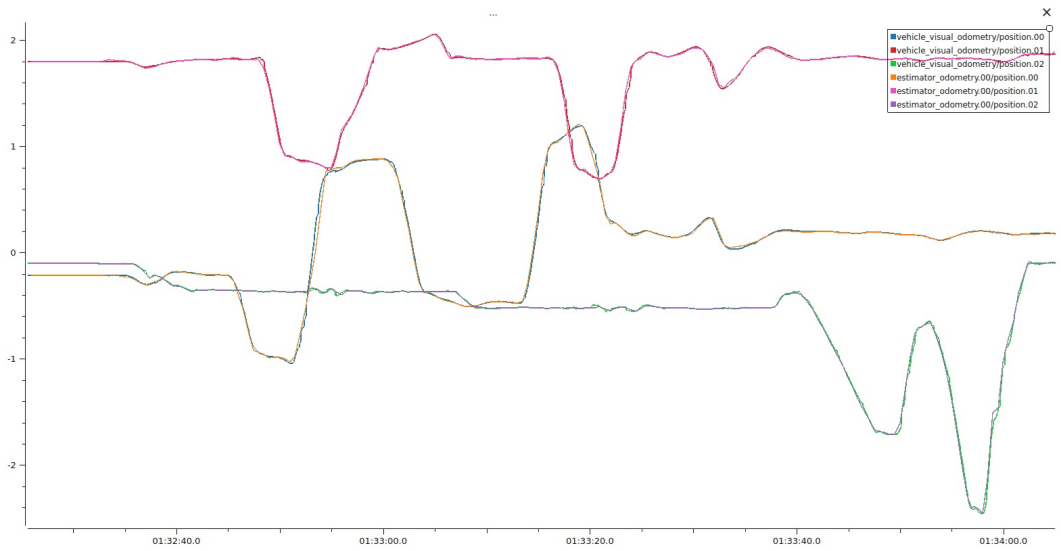

Figure 7.2: Comparison between motion-capture-based external odometry and PX4 estimator position signals during the real flight test

7.4.2 Estimator Fusion Behavior

Once the external odometry stream is injected, EKF2 is expected to accept and fuse it as an aiding source [14, 16]. A practical indicator of correct fusion is that the estimator output follows the external reference smoothly throughout the mission phases, without unstable corrections, repeated resets, or clearly inconsistent responses.

In this work, fusion behavior was evaluated by comparing:

- the injected external odometry, representing the aiding input delivered to PX4;
- the estimator odometry output, representing the EKF2 state used by the control stack;
- the motion-capture-derived pose used as the external localization source.

The objective of this comparison is not to claim perfect tracking, but to show that the estimator operates in a stable regime and that the aiding source is consistent with the expected vehicle motion. As shown in Figure 7.3, the estimator output evolves smoothly and remains qualitatively aligned with the external odometry input during the main phases of the mission. This behavior is compatible with correct fusion of the external aiding source and supports the readiness of the integrated pipeline for closed-loop offboard flight.

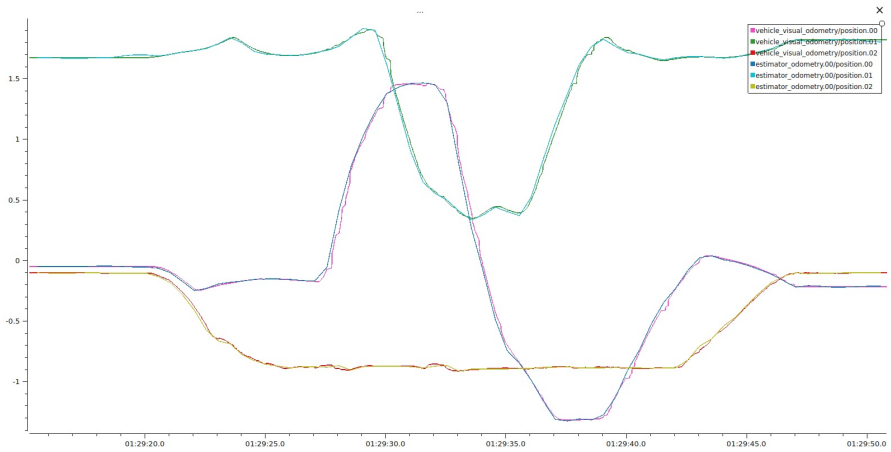

Figure 7.3: Injected external odometry and PX4 estimator output during the mission

7.4.3 Trajectory Tracking Under Offboard Control

The mission also validates the control interface: the offboard controller sends setpoints and the vehicle follows them while relying on the Vicon-aided state estimate. From the point of view of this thesis, the most important result is not centimeter-level path accuracy, but the fact that the vehicle can execute the commanded phases in sequence—takeoff, trajectory motion, and landing—without losing estimator consistency or control continuity.

A compact way to summarize this behavior is to visualize the reconstructed flight path. Even when interpreted qualitatively, the trajectory gives immediate information about whether the vehicle completed the intended mission structure and whether the resulting motion remained physically plausible and continuous.

Figure 7.4 offers a qualitative view of the executed trajectory during the indoor mission. The path is sufficiently structured to show that the vehicle completed the commanded motion phases under offboard control, while remaining compatible with the estimator behavior observed in the time-series plots. In this sense, the trajectory view complements the signal-level comparisons by showing the overall spatial outcome of the experiment.

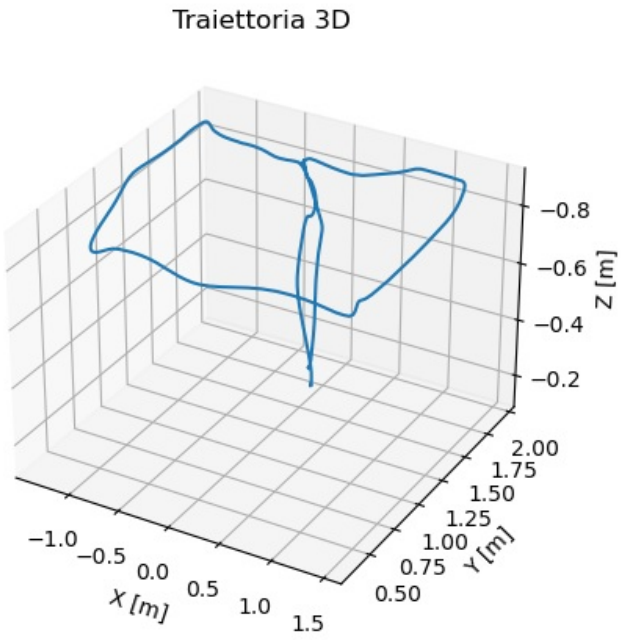

Figure 7.4: Qualitative 3D view of the executed mission trajectory during indoor offboard flight

7.5 Discussion of Results and Limitations

7.5.1 Reliable Outcomes

The staged integration methodology described in Chapter 6 proved effective in reaching repeatable experiments. In particular:

- validating the motion-capture stream first avoided spending time on downstream debugging when the input itself was not yet trustworthy;
- yaw-consistency checks were an efficient diagnostic for frame-convention mistakes;
- the combined use of live monitoring in PlotJuggler and offline rosbag analysis reduced the need for blind parameter adjustments.

A further practical outcome is that repeatability depended less on "remembering the right commands" and more on enforcing a stable preparation workflow. Once the readiness checks were applied consistently before each session, the transition from infrastructure validation to real flight became significantly more predictable.

7.5.2 Limitations

This experimental demonstration focuses on proof of functionality rather than exhaustive performance characterization. Therefore:

- the mission profile is intentionally simple and short, in order to keep debugging and repeated trials manageable;
- the evaluation is mainly based on signal coherence, estimator stability, and mission completion, rather than on a large statistical dataset;
- additional tuning of estimator parameters and noise models could further improve tracking accuracy, but this was outside the scope of the core integration objective.

These limitations do not reduce the main value of the result, which lies in demonstrating that the full indoor-autonomy pipeline can be configured, validated, and exercised successfully in a real experimental environment. At the same time, they define a clear boundary between the contribution of this thesis and possible future work on performance optimization and more advanced control scenarios.

7.6 Secondary Platform Configuration

A second UAV platform was also configured with the same software stack, including the Ubuntu/ROS 2 environment, the Micro XRCE-DDS bridge, and the motion-capture integration pipeline. This activity served as a practical transfer test for the deployment methodology developed during the thesis.

Although real experimental flights on the second platform are not included in the present work, the configuration process verified that the main setup logic can be reproduced on another vehicle, including package availability, bridge setup, and general parameter alignment. The absence of real-flight results on the second platform is not due to an architectural limitation of the proposed workflow, but to practical constraints of the experimental facility: during the final phase of the work, the indoor flight area was temporarily inaccessible because of ongoing works in the Panetti laboratory of the Department.

For this reason, the contribution of the thesis is not limited to a single “working prototype”, but also includes a reusable integration procedure that can be applied to additional UAV platforms. At the same time, the full experimental validation reported in this thesis refers to the primary UAV only.

7.7 Summary

This chapter presented the experimental demonstration of indoor autonomy on a single UAV using Vicon-based localization. The results show that the full pipeline—from motion capture to ROS 2 conversion, PX4 external odometry injection, EKF2 fusion, and offboard control—can be executed in a coherent and repeatable manner under real indoor conditions. In addition to validating the main platform, the work also showed that the same deployment logic can be transferred to a second UAV configuration. The next chapter concludes the thesis and summarizes the main contributions and future extensions.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

This thesis addressed the configuration and experimental validation of a PX4-based UAV for indoor autonomous flight using Vicon motion capture as an external localization source. The main objective was to build a reliable end-to-end pipeline that delivers motion-capture pose information to PX4 as external odometry, ensuring consistency in both reference frames and time, and enabling autonomous missions through a ROS 2 offboard control interface.

A simulation baseline was first adopted to validate the software stack and the PX4–ROS 2 communication chain in a controlled environment (Chapter 5). This step also included a multi-UAV scenario with a fake Vicon pipeline, used to verify scalability and instance separation before interacting with the real facility.

The methodology was then transferred to the indoor flight cage, where the real Vicon system provided the external pose stream. The experimental demonstration on a single UAV confirmed that the complete workflow can be executed in a repeatable and technically coherent way: motion-capture acquisition, ROS 2 integration, frame conversion, external odometry injection, EKF2 fusion, and offboard mission execution (Chapter 7). Beyond achieving a working flight, the work emphasized engineering aspects that are essential for dependable experiments, such as structured deployment, systematic validation, and observability-driven debugging.

A secondary UAV platform was also configured using the same software and integration logic, confirming the transferability of the proposed workflow beyond the main test vehicle. However, real-flight validation on the second platform could

not be completed within the thesis timeline because the indoor flight facility became temporarily inaccessible due to ongoing works in the Panetti laboratory of the Department.

Overall, the thesis demonstrates that reliable indoor autonomous flight is not only a matter of estimator configuration or controller design, but also of disciplined system integration. In this sense, the main contribution of the work lies in defining and validating a practical methodology that connects motion-capture infrastructure, ROS 2 middleware, PX4 state estimation, and offboard control into a coherent and experimentally verified workflow.

8.2 Main Contributions

The contributions of this work can be summarized as follows:

- **End-to-end indoor autonomy pipeline:** a complete integration chain from Vicon/VRPN pose to PX4 external odometry and EKF2 fusion, enabling autonomous indoor missions through ROS 2 offboard control.
- **Frame-consistent pose injection:** a conversion and packaging layer that enforces PX4 conventions (world NED and body FRD), supported by practical validation methods based on yaw/heading consistency checks and signal coherence across the pipeline.
- **Time-base coherence and delay awareness:** a LAN-based timing strategy enabling coherent timestamp handling, complemented by logging-based delay characterization to support repeatable estimator fusion.
- **Observability-driven integration workflow:** a unified monitoring and data collection approach combining live visualization and offline logging, which reduced blind trial-and-error iterations during integration.
- **Reproducible deployment procedure:** a service-based launch and validation workflow on the companion-computer side, supporting repeatable startup, troubleshooting, and log collection across experimental sessions.
- **Transferability across platforms:** the software stack and deployment methodology were also applied to a secondary UAV platform, verifying practical portability of the configuration workflow, even though real-flight validation on the second UAV could not be completed within the thesis timeframe.

8.3 Multi-UAV Validation in Simulation

In addition to the single-UAV experimental validation, the same integration principles were exercised in simulation in a two-UAV scenario using a fake Vicon pipeline (Chapter 5). The purpose of this activity was methodological: to validate scalability aspects—running multiple PX4 instances, preserving separation between communication paths, and monitoring multi-vehicle state streams—in a controlled environment before considering real multi-agent experiments.

Although limited to simulation, this baseline is relevant because it shows that the proposed architecture is not restricted to a single vehicle. The same building blocks—external state ingestion, estimator fusion, and offboard command interfaces—can be replicated consistently across multiple agents, providing a useful starting point for future coordinated multi-UAV developments.

8.4 Limitations

The work presented in this thesis prioritizes correctness and reliability of the integration pipeline. As a consequence:

- the experimental dataset is intentionally limited to representative demonstration flights, rather than a large statistical campaign;
- performance depends on estimator configuration choices (e.g., noise models, delays, and aiding weights) and could be further refined for higher-precision tracking;
- the multi-UAV scenario is validated in simulation as a controlled baseline; extending it to real indoor coordinated flight requires additional safety layers and coordination logic beyond the scope of this work;
- real-flight validation on the secondary UAV platform could not be completed because the indoor flight facility became temporarily unavailable due to ongoing works in the Panetti laboratory of the Department.

These limitations also define the scope of the contribution: the thesis is centered on integration readiness and experimental proof of functionality, rather than on exhaustive performance optimization.

8.5 Future Work

Several extensions are natural continuations of this work:

- **Broader experimental campaign:** repeat flights across different trajectories and speeds, collecting quantitative tracking metrics and estimator diagnostics to characterize repeatability and accuracy.
- **Robustness to disturbances:** evaluate trajectory tracking under controlled airflow disturbances and refine estimation/control strategies accordingly, including parameter sensitivity analysis.
- **Multi-UAV coordination:** move from the two-UAV simulation baseline toward coordinated behaviors, including collision avoidance, formation control, and shared timing/communication assumptions.
- **Automation and tooling:** further improve deployment scripts, logging automation, and post-processing pipelines to reduce setup time, increase observability, and standardize experiment execution.
- **Secondary-platform experimental completion:** complete the real-flight validation of the second UAV platform once the indoor facility becomes fully accessible again, in order to extend the current transferability result from configuration readiness to experimental verification.

In conclusion, this work provides a concrete foundation for future developments in indoor autonomous flight based on PX4, ROS 2, and external motion-capture localization. The resulting methodology is not only useful for the specific platform considered in the thesis, but also for broader UAV integration activities in which reliability, repeatability, and validation discipline are essential.

Bibliography

[1] *PX4 Vision Autonomy Development Kit*. PX4 Autopilot Documentation. URL: https://docs.px4.io/main/en/complete_vehicles_mc/px4_vision_kit (cit. on pp. 5, 7, 8, 10, 11).

[2] *Holybro Pixhawk 6C*. PX4 Autopilot Documentation. URL: https://docs.px4.io/main/en/flight_controller/pixhawk6c (cit. on p. 6).

[3] *Optical Flow*. PX4 Autopilot Documentation. URL: https://docs.px4.io/main/en/sensor/optical_flow (cit. on p. 8).

[4] *PMW3901-Based Flow Sensors*. PX4 Autopilot Documentation. URL: https://docs.px4.io/main/en/sensor/pmw3901 (cit. on p. 8).

[5] *Distance Sensors (Rangefinders)*. PX4 Autopilot Documentation. URL: https://docs.px4.io/main/en/sensor/rangefinders (cit. on p. 9).

[6] *PM07 Quick Start Guide*. Holybro Documentation. URL: https://docs.holybro.com/power-module-and-pdb/power-module/pm07-quick-start-guide (cit. on p. 10).

[7] *QGroundControl User Guide*. QGroundControl Documentation. URL: https://docs.qgroundcontrol.com/master/en/qgc-user-guide/index.html (cit. on pp. 13, 31, 39).

[8] *Vicon Vero Technical Information*. Vicon. URL: https://www.vicon.com/hardware/cameras/vero/#technical-information (cit. on pp. 16, 17, 45, 54).

[9] Russell M. Taylor II, Thomas C. Hudson, Andreas Seeger, Hans Weber, Adam Juliano, and Arie T. Helser. *VRPN: A Device-Independent, Network-Transparent VR Peripheral System*. Proceedings of the ACM Symposium on Virtual Reality Software and Technology. 2001 (cit. on pp. 17, 23, 29, 46).

[10] *ROS 2 Jazzy Documentation*. Open Robotics. URL: https://docs.ros.org/en/jazzy/index.html (cit. on pp. 17, 26, 45, 46).

[11] *chrony Documentation*. chrony Project. URL: https://chrony-project.org/documentation.html (cit. on pp. 19, 49).

[12] *PX4 Architectural Overview*. PX4 Autopilot Documentation. URL: https://docs.px4.io/main/en/concept/architecture (cit. on p. 23).

[13] Steve Macenski, Tully Foote, Brian Gerkey, William Lalancette, and William Woodall. «Robot Operating System 2: Design, architecture, and uses in the wild». In: *Science Robotics* 7.66 (2022). DOI: 10.1126/scirobotics.abm6074 (cit. on pp. 23, 26, 45).

[14] *Using Vision or Motion Capture Systems for Position Estimation*. PX4 Autopilot Documentation. URL: https://docs.px4.io/main/en/ros/external_position_estimation (cit. on pp. 23, 24, 29, 45, 47, 48, 52, 57).

[15] *uXRCE-DDS (PX4-ROS 2/DDS Bridge)*. PX4 Autopilot Documentation. URL: https://docs.px4.io/main/en/middleware/uxrce_dds (cit. on pp. 23, 27–29, 31, 34, 51).

[16] Paul Riseborough. *PX4 State Estimation*. PX4 Developer Summit presentation. 2020. URL: https://px4.io/wp-content/uploads/2020/07/Paul-Riseborough-PX4-state-estimation-update.pdf (cit. on pp. 24, 57).

[17] *Parameter Reference*. PX4 Autopilot Documentation. URL: https://docs.px4.io/main/en/advanced_config/parameter_reference (cit. on pp. 24, 45, 47, 48, 52).

[18] *ROS 2 User Guide*. PX4 Autopilot Documentation. URL: https://docs.px4.io/main/en/ros2/user_guide (cit. on pp. 28, 34, 51).

[19] *PlotJuggler*. GitHub Repository. URL: https://github.com/facontidavide/PlotJuggler (cit. on pp. 29, 55).

[20] *Gazebo Simulator*. Gazebo. URL: https://gazebosim.org/ (cit. on p. 34).