Robotics Autonomous Systems

Final Project Report

Academic Year 2025-2026

Autonomous Drone: Vision Control

Mechanical Design, State Estimation (EKF), and AI-Driven Control

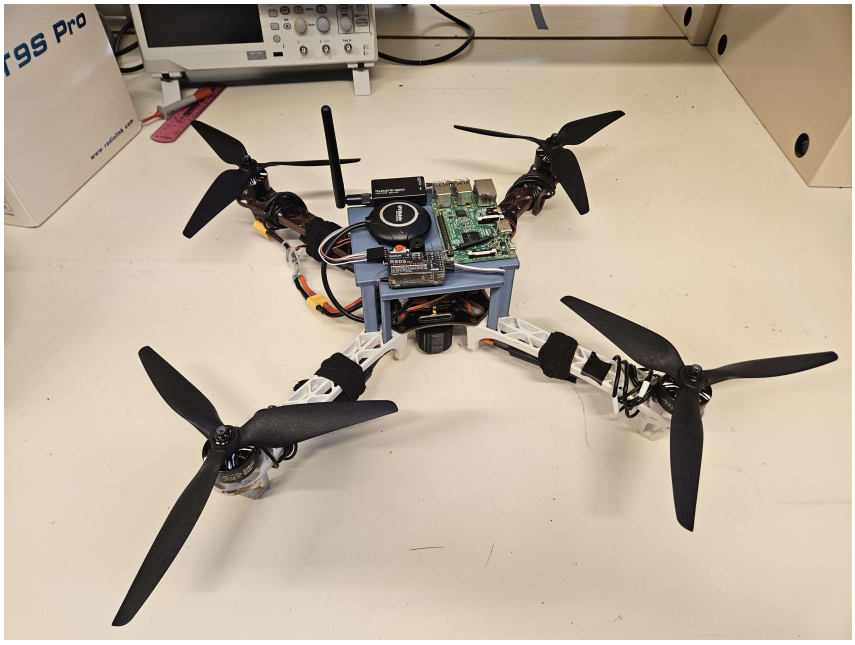

Figure 1 – Final assembled prototype featuring Pixhawk, Raspberry Pi, and Vision Camera.

Students: Rémi GUZZI

and

Daniel WARTSKI NARANJO

Ecole Polytechnique Universitaire de Nice Sophia-Antipolis, Electronics Department
1645 route des Lucioles, Parc de Sophia Antipolis, 06410 BIOT

Table of Contents

1 Introduction

This final year project focuses on the design, implementation, and validation of an autonomous quadrotor drone capable of interacting with its environment through computer vision. In the context of rapidly evolving autonomous systems, the ability to navigate and react to high-level commands without a human pilot is crucial.

The project addresses two main technical challenges:

1. **Robust State Estimation:** Developing a 12-state Extended Kalman Filter (EKF) to fuse noisy data from 10 different sensors (GPS, IMU, Magnetometer) into a coherent state estimate.

2. **Vision-Based Control:** Implementing a high-level control layer using Artificial Intelligence (YOLOv11) to recognize specific human gestures and translate them into flight commands (Landing, Hovering, Translation).

This report details the hardware architecture, the custom mechanical design using CAD, the mathematical modeling of the EKF, simulation results, and finally, the real-world flight tests.

2 Global and Hardware Architecture

To ensure both flight stability and computational power for image processing, we adopted a distributed architecture separating critical real-time tasks from cognitive tasks.

2.1 Distributed System

- **Low-Level (Flight Controller):** A **Pixhawk** handles sensor data acquisition (IMU), motor mixing, and stabilization loops running at high frequency (400Hz).
- **High-Level (Companion Computer):** An onboard **Raspberry Pi 3** manages video streaming and communicates with the ground station via MAVLink over Wi-Fi.
- **Ground Station (GCS):** A high-performance laptop runs the AI inference (YOLOv11) to detect gestures and sends high-level velocity setpoints back to the drone.

Figure 2 – Bottom view of the drone showing the ESC integration, Power Module, and the realsense camera mount.

2.2 Wiring diagram

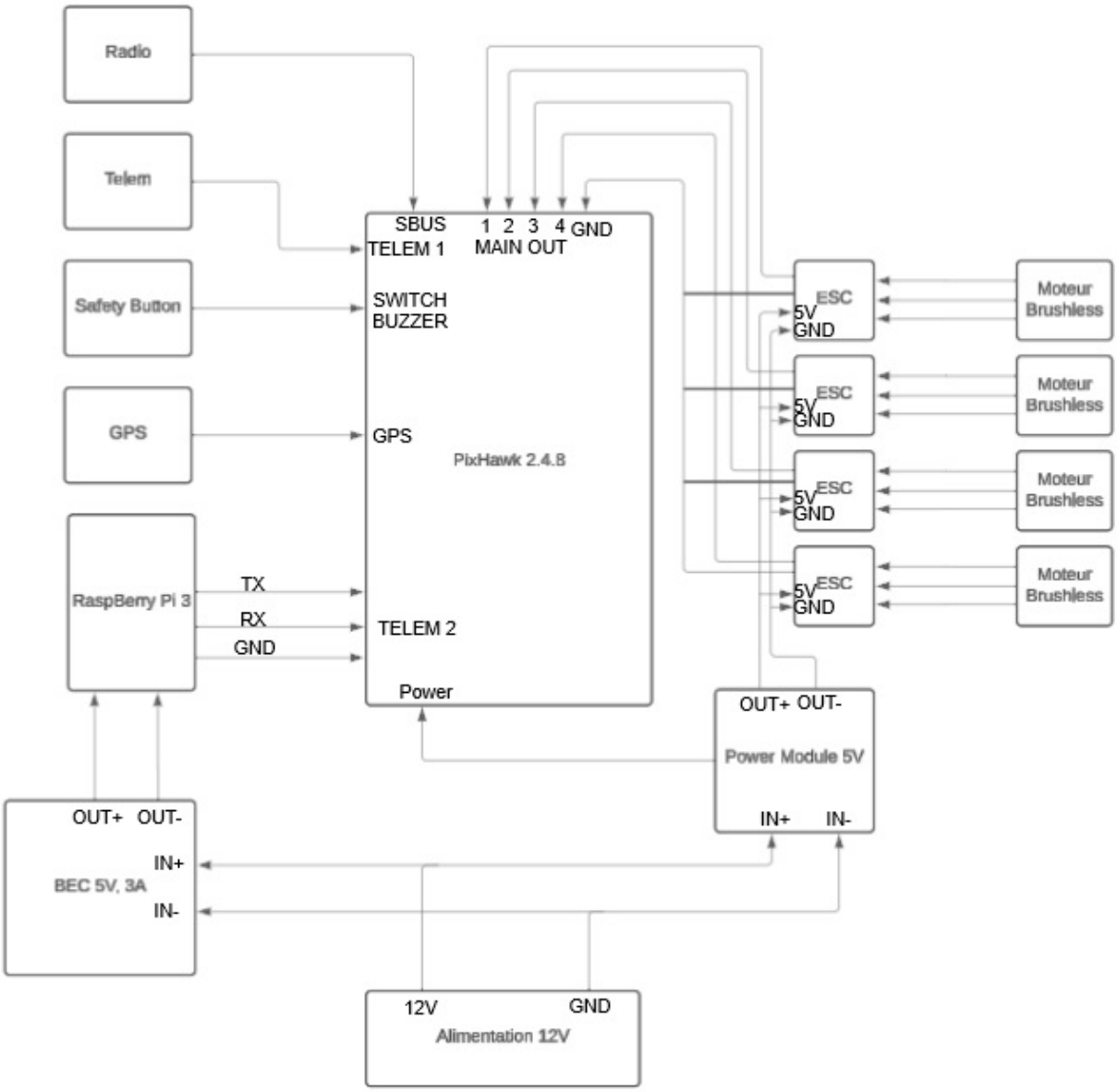

Figure 3 – Wiring diagram

3 Mechanical Design and Integration

A significant portion of the project involved adapting a generic frame to house our specific avionics stack. We modeled custom parts in Fusion 360 and manufactured them using 3D printing (PLA).

3.1 Frame and Propulsion

We utilized a standard F450 quadrotor frame. It is equipped with brushless motors and 9-inch propellers, providing sufficient lift for the additional payload (RPi, Camera, Battery).

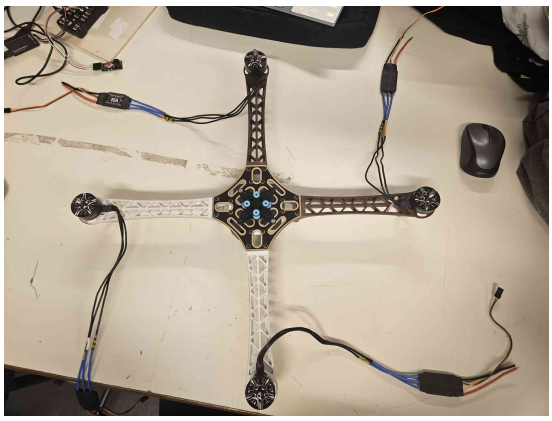

Figure 4 – Bare frame with motors installed.

Figure 5 – Manual soldering of the Power Distribution Board (PDB) to ensure reliable connections.

3.2 CAD Design vs. Realization

To isolate the sensitive IMU sensors from motor vibrations and to securely mount the vision system, custom mounts were engineered.

3.2.1 Flight Controller Mount (Pixhawk)

A vibration-dampened housing was designed for the Pixhawk. This reduces mechanical noise in the accelerometer data, which is critical for the EKF performance.

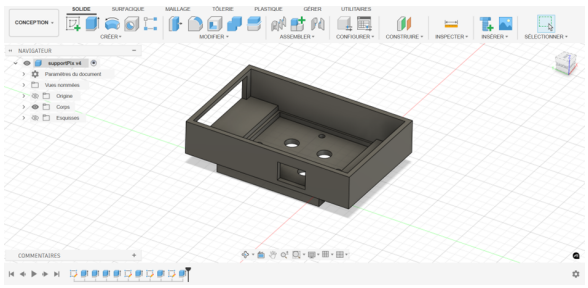

Figure 6 – CAD model of the Pixhawk mount.

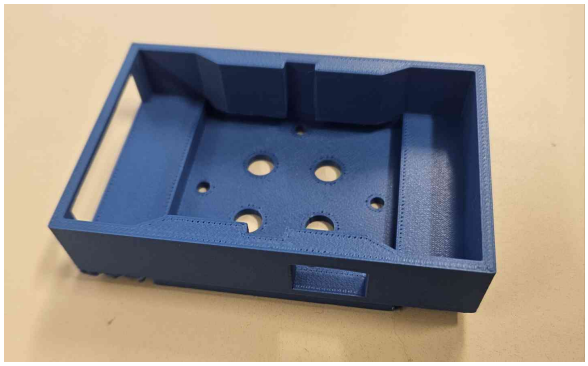

Figure 7 – 3D printed part (PLA) integrated on the frame.

3.2.2 Camera Mount

The camera is mounted beneath the chassis, angled forward and down. This specific field of view allows the drone to detect ground markers for landing and gestures from an operator standing in front of it.

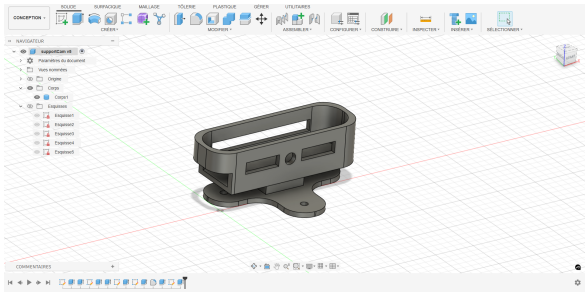

Figure 8 – CAD design of the Camera support.

Figure 9 – Realized mount attached to the bottom plate.

3.3 Final Assembly and Stacking

Due to limited space on the F450 frame, we designed a central stabilization support acting as an intermediate "stack". This allows the Flight Controller to sit centrally while the Raspberry Pi and GPS are mounted on top, optimizing the Center of Gravity (CoG).

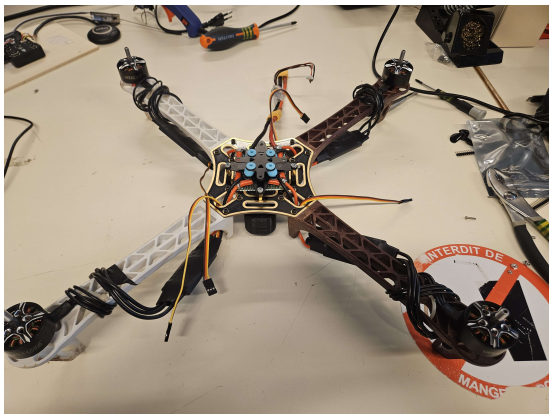

Figure 10 – Central stack support installed over the PDB.

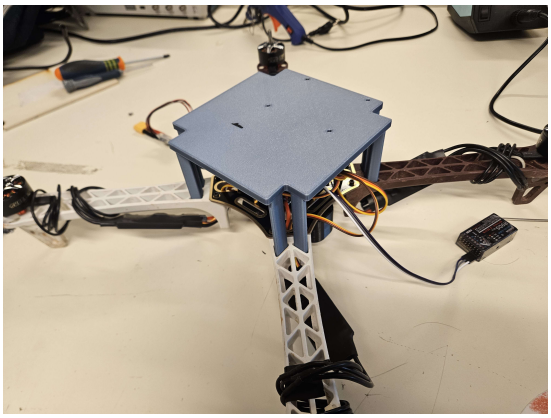

Figure 11 – Top plate installed, ready for the Raspberry Pi.

4 State Estimation: Extended Kalman Filter (EKF)

To obtain a reliable state estimate despite noisy raw sensor data (GPS jitter, accelerometer drift), we implemented a 12-state Extended Kalman Filter.

4.1 State Vector and Dynamic Model

The state vector $X \in \mathbb{R}^{12}$ describes the complete kinematics of the drone. To ensure mathematical coherence and avoid singularities, we defined it as:

$$X = \begin{bmatrix} x & y & z & v_x & v_y & v_z & \phi & \theta & \psi & \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T \tag{1}$$

Instead of using a black-box model, we derived the physical equations of motion. The continuous-time process model $\dot{X} = f(X,U)$ used for the prediction step is:

$$\begin{aligned}
\dot{x} &= v_x, \quad \dot{y} = v_y, \quad \dot{z} = v_z \\
\dot{v_x} &= \frac{U_1}{M}(\sin\theta\cos\phi\cos\psi + \sin\phi\sin\psi) \\
\dot{v_y} &= \frac{U_1}{M}(\sin\theta\cos\phi\sin\psi - \sin\phi\cos\psi) \\
\dot{v_z} &= \frac{U_1}{M}(\cos\phi\cos\theta) - g \\
\ddot{\phi} &= \frac{J_y - J_z}{J_x}\dot{\theta}\dot{\psi} + \frac{L}{J_x}U_2 \\
\ddot{\theta} &= \frac{J_z - J_x}{J_y}\dot{\phi}\dot{\psi} + \frac{L}{J_y}U_3 \\
\ddot{\psi} &= \frac{J_x - J_y}{J_z}\dot{\phi}\dot{\theta} + \frac{1}{J_z}U_4
\end{aligned} \tag{2}$$

4.2 The Yaw Observability Challenge

A major issue encountered during simulation was the unobservability of Yaw (ψ) when relying solely on the accelerometer. While gravity provides an absolute reference for Roll and Pitch, the accelerometer is "blind" to rotation around the Z-axis.

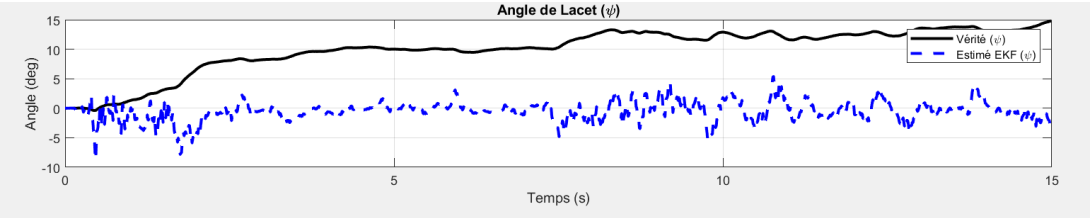

Figure 12 – Yaw drift without magnetometer correction: The simulated drone spins uncontrollably as the error integrates.

By integrating the Magnetometer as a 10th measurement source in the update step, we provided an absolute heading reference, stabilizing the estimate.

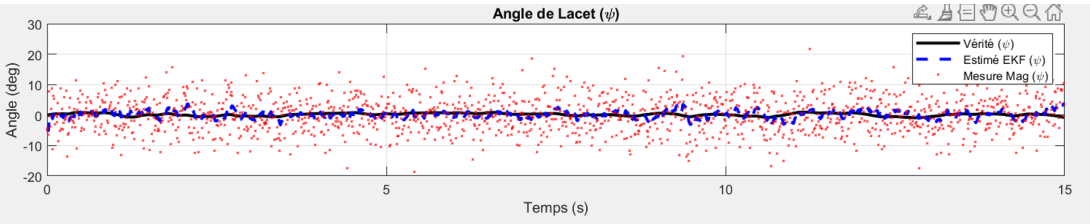

Figure 13 – Stable Yaw estimation (Blue) achieved by fusing Magnetometer data (Red).

5 Simulation and Theoretical Validation

Before attempting real flight, the algorithm was validated in MATLAB using a closed-loop simulation with a cascading PID controller.

5.1 Trajectory Tracking

We simulated a complex flight scenario: Take-off → Backward flight → Lateral flight → Landing.

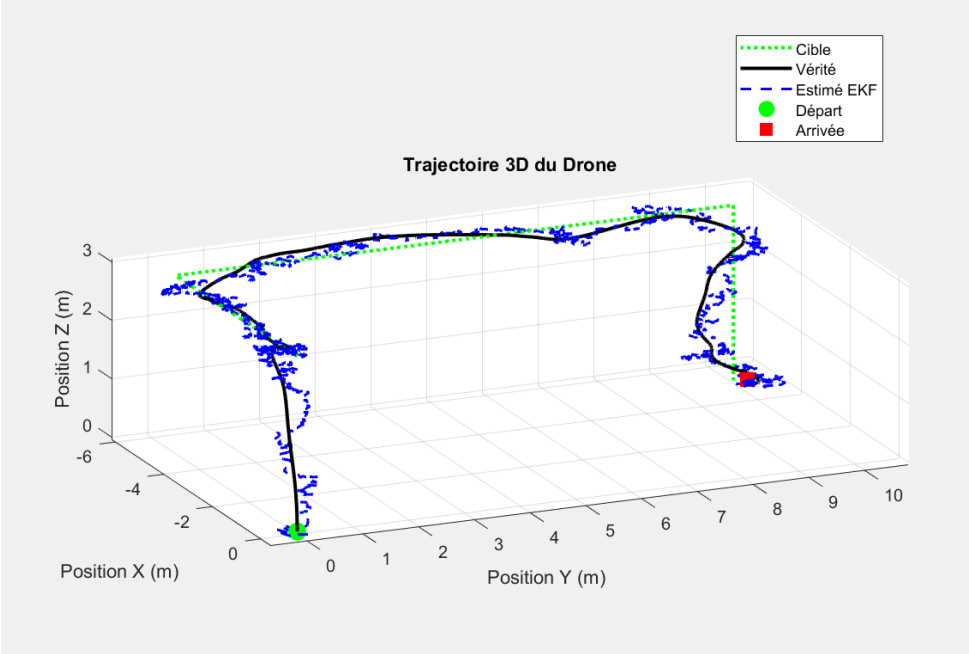

Figure 14 – 3D Simulated Trajectory: The EKF estimate (Blue) tightly tracks the Ground Truth (Black).

5.2 Filter Consistency (3-Sigma Analysis)

To verify that the filter correctly estimates its own uncertainty, we analyzed the estimation error against the covariance bounds ($\pm 3\sigma$). The error remained within bounds, proving the filter is consistent.

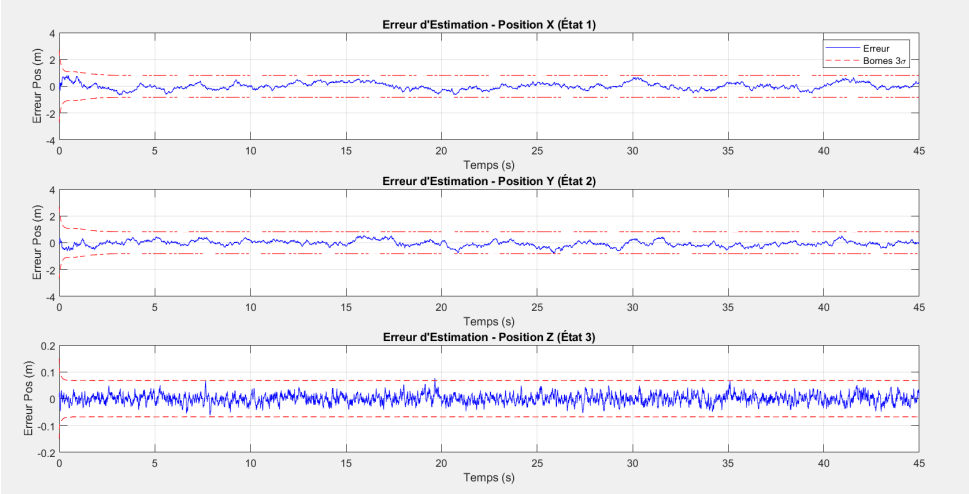

Figure 15 – Position estimation errors vs. 3-sigma bounds.

6 Implementation and Flight Tests

After assembly, we proceeded with configuration using QGroundControl and conducted the first flight tests.

6.1 Calibration and PID Tuning

Sensor calibration is critical. We performed a 6-axis accelerometer calibration and a magnetometer calibration to account for soft/hard iron interference.

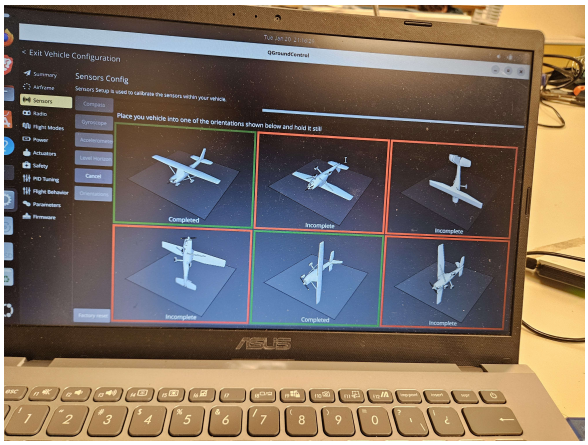

Figure 16 – Accelerometer calibration process.

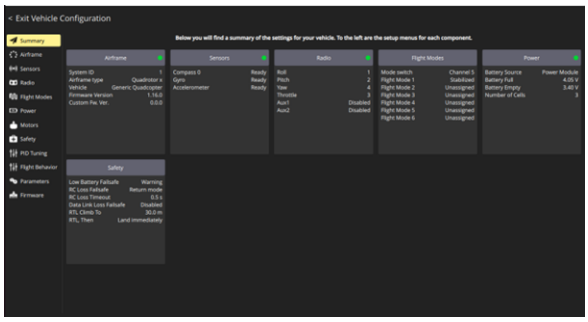

Figure 17 – Sensor status check (All Green).

PID gains were manually tuned. We adjusted the P-gain for responsiveness and the D-gain to dampen oscillations caused by the frame flexibility.

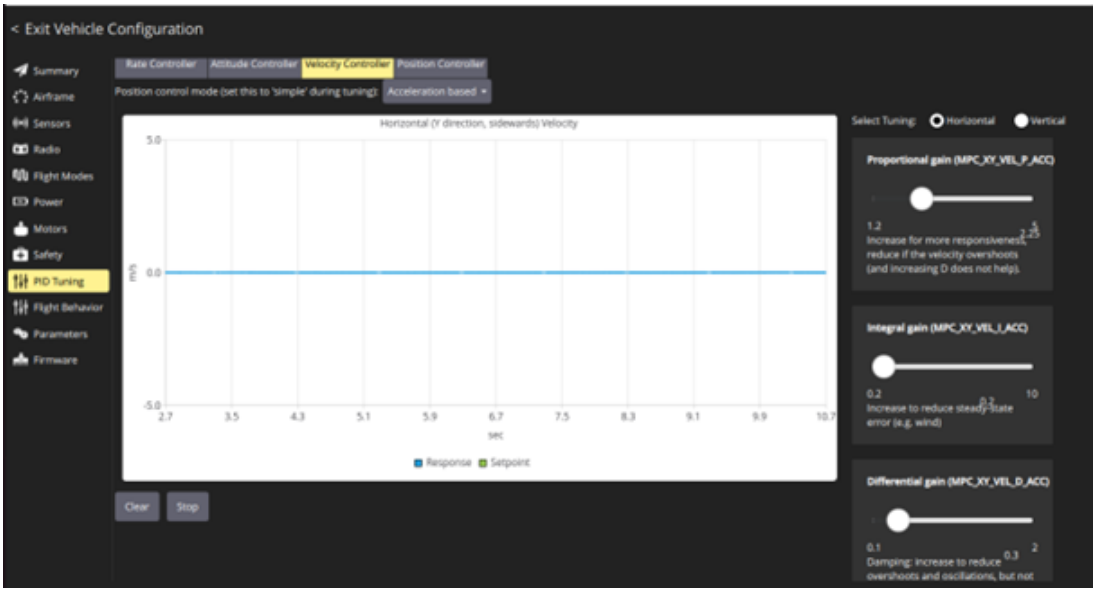

Figure 18 – Velocity Controller PID tuning interface in QGroundControl.

6.2 First Flight Demo

The drone successfully achieved a stable hover in the laboratory, validating the mechanical build and the propulsion chain.

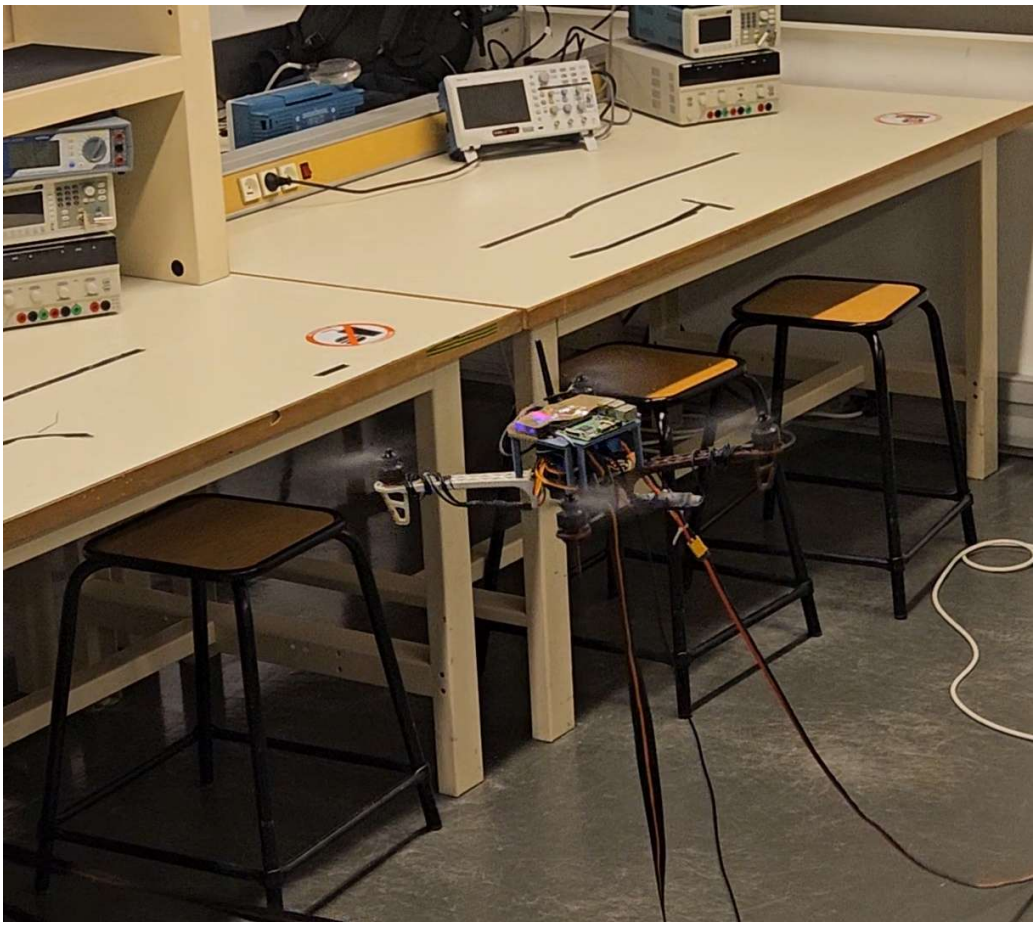

Figure 19 – First stable hover flight in the lab.

Flight Demonstration Video: Click here to watch the flight video

7 Artificial Intelligence: Vision Control

For autonomous control, we trained a lightweight YOLOv11 model to recognize geometric shapes and gestures.

7.1 Control Logic

Each detected class triggers a specific flight mode via MAVLink:

- **Circle:** Landing Sequence.
- **Cross:** Emergency Hover (Position Hold).
- **Parallel:** Forward Translation.

7.2 Detection Results

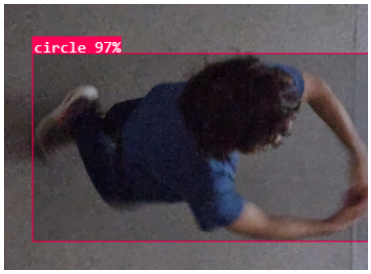

Figure 20 – Circle (Land).

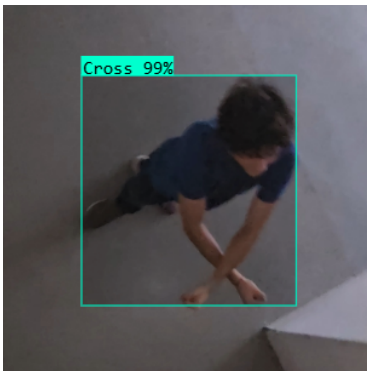

Figure 21 – Cross (Hover).

Figure 22 – Parallel (Move Fwd).

As shown in Figure 22, the "Parallel" gesture (arms extended forward) is detected with 98% confidence. This output is processed by the Ground Station to send a positive Pitch command to the drone.

8 Project Budget

The project was completed within a reasonable budget by reusing an existing frame and utilizing school resources. The table below details the actual expenditure versus the estimated total value of the system.

Category	**Component**	**Paid (€)**	**Est. Value (€)**
Flight Controller	Pixhawk 2.4.8	80.00	80.00
Propulsion	Brushless Motors (4x) + ESC 4-in-1	60.00	60.00
Communication	Radio Telemetry Kit (433MHz)	60.39	60.39
Lidar	Laser Distance Module	14.20	14.20
Hardware	Propellers	11.00	11.00
School Property	*Camera (RealSense D435)*	*0.00*	*250.00*
School Property	*Raspberry Pi 3 v1.2*	*0.00*	*75.00*
School Property	*LiPo Battery (4S 5000mAh)*	*0.00*	*45.00*
Recycled	*F450 Frame Kit*	*0.00*	*30.00*
TOTAL		**225.59 €**	**≈ 625.59 €**

Table 1: Detailed project budget: Out-of-pocket costs vs. Total hardware value.

9 Conclusion

We have successfully designed and built a functional autonomous drone platform. From custom CAD modeling to the theoretical validation of the Extended Kalman Filter and the implementation of AI-based control, all key objectives were met. Future work will focus on porting the simulated EKF code directly onto the Pixhawk firmware and conducting outdoor field tests for the vision system.

10 References

References

[1] Elsevier / ScienceDirect. *Scientific Article (Reference S1270963822004990)*. Available at: https://www.sciencedirect.com/science/article/abs/pii/S1270963822004990?via%3Dihub

[2] Roboflow Blog. (2024). *How to Train a YOLOv11 Object Detection Model on a Custom Dataset*. Available at: https://blog.roboflow.com/yolov11-how-to-train-custom-data/