Integration of a Laser Precision Landing System for UAVs in All-Weather and Dynamic Environments

Embedded Electronic System Design

Shuhang Xie, Rui Cheng

DEPARTMENT OF MICROTECHNOLOGY AND NANOSCIENCE

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

Master's thesis 2026

Integration of a Laser Precision Landing System for UAVs in All-Weather and Dynamic Environments

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Microtechnology and Nanoscience
Chalmers University of Technology
Gothenburg, Sweden 2026

Integration of a Laser Precision Landing System for UAVs in All-Weather and Dynamic Environments

Supervisor: Patxi D. Rodriguez Acero, Fluid Dynamics and Thermal Sciences Laboratory
Examiner: Lena Peterson, Microtechnology and Nanoscience

Master's Thesis 2026
Department of Microtechnology and Nanoscience
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: An illustration of an autonomous UAV utilizing active infrared tracking for precision landing in a GPS-denied maritime environment.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2026

Integration of a Laser Precision Landing System for UAVs in All-Weather and Dynamic Environments
Department of Microtechnology and Nanoscience
Chalmers University of Technology

Abstract

Autonomous unmanned aerial vehicle (UAV) landings in GPS-denied and visually degraded environments remain a significant challenge. Standard Global Navigation Satellite System (GNSS) solutions are susceptible to multipath interference near metallic structures and provide absolute rather than relative positioning, making them unsuitable for landing on dynamic platforms. Passive visual sensors fail under low light, fog, and direct glare, and introduce significant processing latency.
This thesis integrates an active infrared precision landing system based on the Sixdof Space Kipa optical sensor, which operates at up to 400 Hz across all lighting conditions. A decentralized control architecture was developed combining a LattePanda Mu companion computer with a Pixhawk 6C flight controller. A custom C++ application implements a three-stage signal processing pipeline (median filter, chi-squared outlier gate, 1D Kalman filter), weather-adaptive proportional-integral-derivative control with gain scheduling, a hierarchical descent governor, and a finite state machine enforcing safe behavior under sensor loss and extreme conditions. Velocity commands are transmitted to the flight controller via the MAVLink protocol.
Simulation-in-the-loop across 24 test configurations demonstrated sub-centimeter mean landing error under calm conditions, *Excellent* classification (landing accuracy within 300 mm) at steady wind speeds up to 15 m/s, and *Excellent* classification at turbulence intensities up to 7. Under a combined worst-case scenario of 15 m/s wind and turbulence intensity 5, all five runs confirmed touchdown with a mean error of 181.4 mm. The end-to-end pipeline latency was measured at 52 ms, satisfying the 75 ms design objective. Physical characterization of the Kipa sensor confirmed sub-3 cm depth accuracy at touchdown range and near-zero position noise. Hardware bench integration was verified through confirmed motor spin-up under algorithmic control.

Keywords: Unmanned Aerial Vehicles, Precision Landing, Active Infrared Tracking, PID Control, MAVLink, GPS-Denied Navigation.

Acknowledgements

First and foremost, we would like to express our deepest gratitude to our supervisor, Patxi D. Rodriguez Acero, for his invaluable guidance, expertise, and continuous support throughout this thesis project at the Fluid Dynamics and Thermal Sciences Laboratory. We are also grateful to our examiner, Lena Peterson, for her insightful feedback and evaluation of our work within the Department of Microtechnology and Nanoscience.

A special thanks goes to Per Larsson-Edefors for his support as the Program Director of the Embedded Electronic System Design master's program, which laid a solid academic foundation for our research.

We would also like to extend our appreciation to our colleagues and peers who provided valuable technical insights and discussions.

Finally, we wish to thank our friends and family for their unwavering encouragement, understanding, and patience throughout our academic journey at Chalmers.

Shuhang Xie, Rui Cheng, Gothenburg, June 2026

Disclaimers

During the preparation of this thesis, generative artificial intelligence (AI) tools were utilized to assist with specific aspects of the research and writing workflow. The application of these tools was strictly limited to the following areas:

- **Language and Style Refinement:** AI was used to proofread, paraphrase, and polish the grammar, flow, and clarity of my original drafted text.
- **Programming and Coding:** AI tools were employed to assist in drafting, optimizing, and debugging code segments and scripts used during the data analysis and development phases.
- **Technical Support:** AI models were consulted to provide operational instructions, troubleshooting guidance, and best practices for utilizing specialized software required for this research.

Contents

List of Figures

List of Tables

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

6DoF	Six Degrees of Freedom
E2E	End-to-End
EKF	Extended Kalman Filter
EKF2	Second-Order Extended Kalman Filter
EMA	Exponential Moving Average
FSM	Finite State Machine
GCS	Ground Control Station
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
IMU	Inertial Measurement Unit
IR	Infrared
LoS	Line-of-Sight
MAVLink	Micro Air Vehicle Link
NED	North-East-Down
OS	Operating System
PID	Proportional-Integral-Derivative
PWM	Pulse Width Modulation
QGC	QGroundControl
RMS	Root Mean Square
RTK	Real-Time Kinematic
RX	Receiver
SDK	Software Development Kit
SITL	Software-In-The-Loop
TCP	Transmission Control Protocol
TX	Transmitter
UART	Universal Asynchronous Receiver-Transmitter
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
USB	Universal Serial Bus

1
Introduction

Autonomous unmanned aerial vehicle's (UAV) use in real-time monitoring, search and rescue, security and surveillance, agriculture, civil infrastructure inspection and emergency service has been expanding quickly and significantly [1]. With its high-speed growth and its versatile nature, UAV technology has also been improving rapidly to satisfy the demands. Though UAVs are nowadays highly reliable in many operations, landing them can still be very challenging [2] which the industry phrases it as the "last meter problem".
Especially, when it comes to landing a drone on a moving platform, this "last meter problem" can be even more challenging due to newly incurred external disturbances and localization errors [3]. Apart from that, there are many missions conducted in GPS-denied (Global Positioning System) environment (eg. around large structures or inside warehouse) which compels the industry to develop new landing techniques such as vision-based landing. However, the environment of landing can sometimes be even more unpredictable and complicated because situations like fog, heavy rain, dark environment are common and can negatively affect certain types of sensors and camera helping land the drone. Consequently, a robust system for UAV's landing is urgently needed to mitigate or perhaps completely solve the environmental and dynamic disturbances.

1.1 Problem Statement

Landing techniques that are currently widely used are standard global navigation satellite system (GNSS) and real-time kinematic (RTK). But they inherently present some limitations. In ideal conditions, they can achieve real-time centimeter-level kinematic navigation. However, when environmental conditions deteriorate—such as poor sky visibility or severe multipath effects near large metal structures—GPS performance degrades, leading to significant variations in the drone's tracking quality [4].
Another solution to avoid using GPS is vision-based landing and often links to the use of fiducial markers (e.g., AprilTags [5]) which is capable of achieving high precision landing but at the same time, inevitably limited by its landing environment. Fiducial markers, in essence, are artificial patterns like QR code or a certain reference pattern for on board camera to recognize and help UAV to ascertain its position. However, those markers often fail in low light conditions since it is passive system (fiducial markers in the landing pad do not actively emit light or send signals to the drone). More importantly, they are images and require image processing for the on

board computer, which can introduce significant latency and make it less effective in dynamically controlling the UAV when it comes to landing in a moving platform.

1.2 Proposed Solution

To overcome the limitations of GPS and vision-based landing in challenging environments, we propose an active infrared precision landing system based on Sixdof Space technology. The whole system comprises an onboard kipa sensor for real-time detecting IR signals, and a ground map on which more than three IR beacons spread. The system can update the UAV's six-degrees-of-freedom (6DoF) position at up to 400 Hz which is suitable for landing in challenging conditions like windy day and moving platform needing very fast feedback for stable control. Instead of using passive markers on the ground, it uses beacons to emit infrared pulses actively on the ground to function robustly from total darkness (0 lx) to direct sunlight (100 000 lx).

1.3 Objectives and Contributions

Our primary objective is to integrate our IR equipment on a multirotor drone and validate the whole landing system is achievable by ensuring the data gathered by the sensor can ultimately turn into commands to control UAV's individual motor. Since a typical moving landing platform (e.g. a ship deck or ground vehicle) moves at frequencies well below 1 Hz, meaning its position changes meaningfully over timescales of seconds. Against that movement, we set up a benchmark for the sufficient real-time update rate.
Specifically, our goals include:

- **Overall system functionality:** Integrate each component of the system to fully realized actual landing ability. Once the sensor collects the data, the system should be able to set drone's motors in motion in accordance with the control algorithm.
- **Control algorithm:** Develop a control algorithm that is robust to fight against adverse weather and environment. Ideally, the control algorithm should land the drone within 0.5 m of the landing position, withstand strong wind up to 7 m/s. The design should refrain from using any GPS data to prove it work in GPS-denied environment.
- **Safety:** Even if there is occasional malfunctions in software or the landing condition is not met, the system should keep the UAV steady and avoid losing control, i.e. causing the drone to crash or uncontrolled flip-over.
- **Latency:** Make sure the latency of the whole workflow, from sensor's data to ultimately sending the control commands, is less than 75 ms.

2
Theoretical Background

2.1 Six-Degree-of-Freedom Tracking

A rigid body in three-dimensional space has six degrees of freedom (6DoF) [6], consisting of three translational degrees of freedom and three rotational degrees of freedom. The translational degrees of freedom describe the position along the X, Y, and Z axes, while the rotational degrees of freedom describe the orientation about these axes, commonly represented by roll, pitch, and yaw.

2.2 Infrared Beacon-Based Tracking

Infrared (IR) beacons can be used as active reference points for determining the position of a mobile platform. By placing multiple beacons at known locations and detecting their emitted signals using a sensor mounted on the platform, the relative position of the platform can be estimated [7]. This principle has also been investigated for autonomous UAV landing, where infrared beacons provide a cooperative reference for landing guidance in environments where conventional positioning systems may be unavailable or unreliable.
In this thesis, an IR beacon-based tracking system is used to provide the UAV with 6DoF measurements during the landing process. The specific tracking hardware and its configuration are described in both Method chapter and Design and Implementation chapter.

2.3 MAVLink Communication Protocol

MAVLink (micro air vehicle link) is a lightweight binary communication protocol [8] designed for communication between unmanned vehicles and external systems. It is commonly used to exchange telemetry, state information, commands, and configuration data between a flight controller, companion computer, and ground control station.
MAVLink communication is message-based. Different message types are defined for different categories of information, including vehicle state, position, velocity, flight mode, and control commands. This allows external systems to communicate with a flight controller using a standardized protocol.
In a UAV system containing a companion computer, MAVLink can be used to transfer high-level commands from the companion computer to the flight controller. The

flight controller can also transmit vehicle-state information back to the companion computer, providing feedback for the high-level control algorithm.
The use of MAVLink therefore allows the high-level control logic and low-level flight-control system to operate as separate components while maintaining a standardized communication interface. The specific MAVLink messages and communication configuration used in this thesis are described in the Method chapter.

2.4 Flight Controller

A flight controller is responsible for the low-level control and stabilization of a UAV. It receives desired motion or attitude commands and uses measurements from onboard sensors to determine the actuator outputs required to achieve those commands.
A typical flight controller contains an IMU (inertial measurement unit), which provides measurements of acceleration and angular velocity. Additional sensors, such as a barometer and magnetometer, may also be used to estimate the vehicle's state. The flight controller combines sensor measurements with an estimation algorithm to obtain an estimate of the UAV's state. State estimation is necessary because individual sensors provide incomplete or noisy measurements. The estimated state can then be used by the flight-control loops to stabilize the vehicle and control its motion.
Extended Kalman filters (EKFs) are commonly used in UAV flight controllers for state estimation. An EKF combines measurements from multiple sensors with a model of the vehicle dynamics to estimate quantities such as position, velocity, and attitude.
In the system considered in this thesis, the flight controller is responsible for the low-level stabilization of the UAV, while higher-level landing control is performed externally. The specific flight-controller hardware and its configuration are described in the Method chapter.

2.5 Sensor Filtering

Sensor measurements are affected by noise and disturbances. In a feedback-control system, noisy measurements can result in unstable or undesirable control behavior because the controller responds not only to the actual motion of the UAV but also to measurement errors.
Filtering can therefore be applied to sensor measurements before they are provided to the control algorithm. Different filtering methods are suited to different types of disturbances. In this thesis, a multi-stage filtering approach consisting of a median filter, a statistical outlier gate, and a one-dimensional Kalman filter is considered.

2.5.1 Median Filter

A median filter is a nonlinear filtering technique commonly used to suppress impulsive noise and isolated outliers. Instead of calculating the arithmetic mean of a set

of measurements, the median filter sorts the measurements and selects the middle value [9].
For a sliding window containing N measurements, the output can be expressed as

$$y_k = \text{median}\left(x_k, x_{k-1}, \ldots, x_{k-N+1}\right), \tag{2.1}$$

where x_k is the current measurement and y_k is the filtered output.
The median filter is particularly effective against isolated spikes because an extreme measurement has limited influence on the median. However, its ability to reject disturbances depends on the size of the sliding window and the duration of the disturbance. A disturbance that persists over a sufficiently large portion of the window may therefore pass through the filter.

2.5.2 Chi-Squared Outlier Detection

Statistical outlier detection can be used to determine whether a new sensor measurement is consistent with previously observed measurements. One approach is to evaluate the measurement using a chi-squared statistical test [10].
Given a set of historical measurements h_i, the sample mean can be calculated as

$$\mu = \frac{1}{N}\sum_{i=1}^{N} h_i, \tag{2.2}$$

while the corresponding variance is

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(h_i - \mu)^2. \tag{2.3}$$

For a new measurement z_k, its normalized squared deviation from the historical mean can be expressed as

$$\chi^2 = \frac{(z_k - \mu)^2}{\sigma^2}. \tag{2.4}$$

A sufficiently large value indicates that the new measurement deviates significantly from the distribution of the recent measurements. A threshold can therefore be used to classify the measurement as either valid or an outlier.

2.5.3 Kalman Filter

The Kalman filter is a recursive state-estimation method used to estimate the state of a dynamic system from noisy measurements. It combines a prediction based on a mathematical model with incoming measurements to produce an updated state estimate.
For a discrete-time linear system, the state and measurement models [11] are given by

$$\mathbf{x}_k = \mathbf{F}_k\mathbf{x}_{k-1} + \mathbf{B}_k\mathbf{u}_k + \mathbf{w}_k, \tag{2.5}$$

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k, \tag{2.6}$$

where $\mathbf{x}_k$ is the system state, $\mathbf{u}_k$ is the control input, $\mathbf{z}_k$ is the measurement, and $\mathbf{F}_k$ and $\mathbf{H}_k$ are the state-transition and observation matrices. The terms $\mathbf{w}_k$ and $\mathbf{v}_k$ represent process and measurement noise, respectively.
The filter operates in two steps. First, the state is predicted using the system model:

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}_k \hat{\mathbf{x}}_{k-1|k-1} + \mathbf{B}_k \mathbf{u}_k. \tag{2.7}$$

When a new measurement becomes available, the prediction is corrected using the Kalman gain:

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \left(\mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1} \right). \tag{2.8}$$

The Kalman gain determines the relative influence of the prediction and the measurement based on their respective uncertainties. The specific state model and filter parameters used in this thesis are described in the Method chapter.

2.6 Ground Control Station

A ground control station (GCS) is a software application used to communicate with and monitor a UAV flight controller. GCS software can provide information about vehicle state, flight mode, position, and system status while also allowing the user to configure flight-controller parameters.
GCS applications commonly communicate with the flight controller using MAVLink. Depending on the flight-control software, a GCS can also be used for tasks such as sensor calibration, parameter configuration, mission planning, and flight-mode selection.

3
Method

3.1 The Sixdof Space Falcon Tracking System

Sixdof Space Falcon is the specific product type we are using to provide precise and real-time 6DoF data [12]. By attaching the Kipa sensor (optical sensor) to the drone[13], the Sixdof Space Falcon is able to collect the 3D coordinates (X, Y, Z) and orientation angles (Roll, Pitch, Yaw) of a UAV. With centimeter-level accuracy and an angle detection with less than 0.5° deviation, the system is suitable for precision drone landing[14].

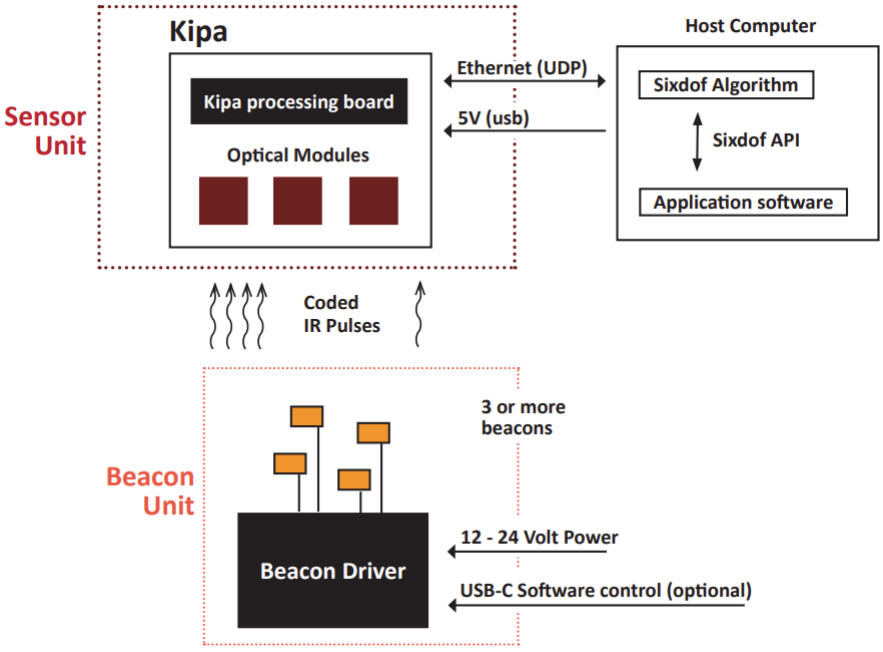

Figure 3.1: Sixdof Architecture

3.1.1 Hardware Architecture

The Falcon hardware development kit consists of three crucial components which are the beacons, the beacon driver board and the Kipa sensor (see Fig. 3.1) [13]. The beacons and its driver board are set on the ground to provide spatial references to the Kipa sensor[14].
There are a wide range of beacon types to choose between. However, they are

fundamentally three types, which are low-power (0.5 W) beacons for close tracking (up to 7.6 m), high-power (4.6 W) HYF beacons for tracking up to 23.5 m and even very-high-power (144 W) HYFS16 beacons for tracking up to 100 m [15]. In our case, we use 4 low-power beacons (see Fig. 3.2) for close-range tracking and 4 high-power HYF beacons (see Fig. 3.3) for tracking over 10m height.

These beacons are powered by the beacon driver board. Using different driver board means different power consumption and different maximum tracking distance even for the same type of beacon. The driver board powers our low-power beacons at 0.5 W with maximum tracking distance of 7.6 m and our high-power beacons at 4.6 W with maximum tracking distance of 23.5 m. The receiving end of Falcon system is the Kipa sensor, which is equipped with optical filters to eliminate noise caused by ambient light[14].

Figure 3.2: Low Power Beacon

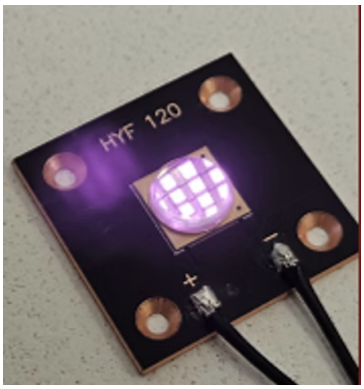

Figure 3.3: HYF Beacon

3.1.2 Falcon's Advantages

One area where the Falcon system distinguishes itself from vision-based system is that the Kipa sensor directly detects the pulses from the beacons. Although this active tracking requires a power source on the landing pad, it accelerates the whole tracking by not needing to process pixel-based image which could be very slow. It also requires less ambient light since the beacons emit IR signals on their own.

The Falcon system is also inherently better than a GPS-based landing procedure because it works in GPS-denied environment. After the Kipa sensor receives the IR signals, it streams the compressed optical data through User Datagram Protocol (UDP) to the host computer, where the Falcon's software will calculate the 6DoF pose in 2.5 ms (400 Hz refresh rate) [14]. Operating at this refresh rate, which significantly surpasses the typical 10-20 Hz refresh rate [16] of RTK or GNSS based landing, the Falcon ensures near-zero latency which is crucial for steady control in challenging tasks.

3.2 LattePanda Mu and Lite Carrier Platform

The LattePanda Mu with Lite Carrier is our host computer on-board for processing the raw 6DoF data and ultimately turning them into commands that are recognized by the flight controller. It has a Intel N100 x86 processor running Ubuntu Linux [17]. The Lite Carrier board is equipped with a Gigabit Ethernet port [18] for receiving 6DoF data from the Kipa sensor via UDP. The board also has multiple USB ports that can supply 5 V power to the Kipa sensor. Once the data received, the control algorithm on LattePanda will start processing and sending commands to flight controller (Pixhawk).

3.3 MAVLink and Pixhawk Flight Controller

While LattePanda Mu does the heavy-lifting of our control logic, Pixhawk is responsible for low-level control. MAVLink is the communication protocol to make them work together.

3.3.1 MAVLink Communication Protocol

In our project, MAVLink bridges the LattePanda host computer and the Pixhawk flight controller. That means when the control logic finishes the calculation based on the sensor data and starts generating velocity commands to correct the path of the UAV, all of those commands are packaged into MAVLink messages and then transmitted to the Pixhawk flight controller. Pixhawk will recognize these MAVLink messages and execute the low-level control to spin the motors. Pixhawk also uses the MAVLink protocol to stream back the drone's status so that the control algorithm can make adjustment based on the commands to drone.

3.3.2 Pixhawk 6C

Pixhawk 6C is the flight controller that executes the lowest-level control in our system. It translates the commands sent by our control algorithm, into pulse width modulation (PWM) signals that physically drive the motors on the drone. It contains a STM32H743 ARM Cortex-M7 processor working at 480 MHz and is itself a sensor too with inertial measurement units, a barometer and a magnetometer [19]. It is also equipped with a vibration isolation system that can filter high-frequency mechanical noise from the motors to ensure its sensors are reading correct data during flight. Inside the Pixhawk, there is also an EKF that continuously estimates the state of the drone such as velocity, position and altitude. EKF essentially serves as the 'glue' to combine all the Pixhawk's sensor data and produces accurate information about the drone's status.
Pixhawk also communicates with ground control. In our actual hardware integration, the QGroundControl is used. Ground control can override the UAV's flight mode if necessary, and the Pixhawk is continuously sending the `Armed` status (i.e., motors are enabled and ready to spin) and flight mode information back to our

control algorithm in LattePanda, enabling the algorithm to adjust the generated commands.

Figure 3.4: Pixhawk 6C

3.4 Ground Control Station Software

GCS is the application used for communicating with the flight controller via MAVLink. It plays a vital role in UAV's mission because it not only monitors the flight status and flight mode of the UAV, but also configures the flight controller such as calibrating sensors on the flight controller, overriding the flight mode of the drone, changing the parameters of flight controller, etc. Two different GCS applications are used in this project which are Mission Planner and QGroundControl (QGC).

3.4.1 Mission Planner

Mission Planner is a open-source GCS based on ArduPilot firmware. The key feature we need in our project is its built in software-in-the-loop (SITL) simulation environment [20]. It can simulate a virtual ArduPilot-based flight controller and drone in its software so we can easily simulate our control logic on a PC. It allows our algorithm in LattePanda to directly send velocity commands via MAVLink to control the virtual drone on its software. Mission Planner also sends back the drone's position data as a feedback to our control algorithm without using an actual sensor. This simulation set up allows us to test the functional correctness of our control algorithm without setting up any physical component.

3.4.2 QGroundControl

QGroundControl (QGC) is an open-source GCS applicaiton as well which is compatible with our PX4-based [21] flight controller Pixhawk 6C. We used it during our physical hardware integration and bench test. Essentially, it clearly states the Pixhawk's arm status and its flight mode to the users and allows users to configure the EKF2 (second-order extended Kalman filter) parameters to instruct Pixhawk how to fuse position data from our Kipa sensor to determine the drone's arm status and flight mode.

3.5 Multi-Stage Filtering

In dynamic outdoor environment, optical sensor is susceptible to signal degradation, including impulse noise from rain drops and Gaussian noise from nearby environment. To ensure the control stability, raw positional data must go through the filtering pipeline consisting of a median filter, a chi-squared outlier gate, and a 1D Kalman filter.

3.5.1 Median Filter

The first stage of the filter pipeline addresses impulse noise caused by momentary obstruction that blocks the Kipa sensor. The median filter is a technique that guards the data reading from isolated spikes. For a sliding window of size N, it outputs the middle value [9]:

$$y_k = \text{median}(x_k, x_{k-1}, \ldots, x_{k-N+1}) \tag{3.1}$$

However, it does not protect against sustained bad readings last more than or equal to half of the total frames. We use in total $N = 5$ frames so it cannot protect bad readings last more than 2 frames.

3.5.2 Chi-Squared Outlier Gate

To counter sustained bad readings or multiple spikes in the data that bypass the median filter, we employed a chi-squared outlier gate [10] to evaluate the plausibility of a new measurement z_k coming from the Median Filter. It keeps a history of last 20 validated readings and calculates their sample mean μ and variance σ^2.

$$\mu = \frac{1}{N}\sum_{i=1}^{N} h_i \tag{3.2}$$

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(h_i - \mu)^2 \tag{3.3}$$

where $N = 20$ is the history window size and h_i denotes the i-th validated reading in the history buffer. Then it computes the squared Mahalanobis distance between the newly read data and the mean of the recent 20 validated history data:

$$\chi^2 = \frac{(z_k - \mu)^2}{\sigma^2} \tag{3.4}$$

If χ^2 exceeds a predefined threshold τ (in this case, $\tau = 9.0$ for a 3σ confidence interval), the measurement is rejected, and the filter outputs the last validated data.

3.5.3 1D Kalman Filter

The final stage applies a discrete-time 1D Kalman filter [22] independently to each spatial axis to reduce residual Gaussian noise and produce a velocity estimate alongside the cleaned position output.
The filter has a two-element state vector of position and velocity on a single axis:

$$\mathbf{x}_k = \begin{bmatrix} p_k \\ v_k \end{bmatrix} \tag{3.5}$$

where $p_k \in \mathbb{R}$ denotes the position estimate and $v_k \in \mathbb{R}$ the velocity estimate along a single axis at time step k. Since the Kipa sensor measures only position, velocity is not directly observed but inferred from how all the position changes between two successive callbacks. The filter operates in two steps on every sensor update:
Prediction: It uses a constant-velocity kinematic model, the filter estimates where the drone should be before the new measurement arrives:

$$\hat{p}_{k|k-1} = \hat{p}_{k-1} + \hat{v}_{k-1} \cdot \Delta t \tag{3.6}$$

$$\hat{v}_{k|k-1} = \hat{v}_{k-1} \tag{3.7}$$

where $\hat{p}_{k|k-1}$ and $\hat{v}_{k|k-1}$ are the predicted position and velocity at step k given all information up to step $k-1$, $\hat{p}_{k-1}$ and $\hat{v}_{k-1}$ are the previous estimates, and Δt is the time elapsed since the preceding sensor callback.
Update: Upon receiving a validated measurement z_k from the chi-squared outlier gate, the filter corrects its prediction. The Kalman gain $\mathbf{K}_k$ determines how much weight is given to the new measurement versus the kinematic prediction — a high gain trusts the sensor measurement more, while a low gain relies more on kinematic prediction when the sensor signals are degraded:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}^T(\mathbf{H}\mathbf{P}_{k|k-1}\mathbf{H}^T + R)^{-1} \tag{3.8}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k(z_k - \mathbf{H}\hat{\mathbf{x}}_{k|k-1}) \tag{3.9}$$

where $\mathbf{P}_{k|k-1} \in \mathbb{R}^{2\times2}$ is the predicted error covariance matrix, $\mathbf{H} \in \mathbb{R}^{1\times2}$ is the observation matrix that maps the state vector to the scalar measurement space, $R \in \mathbb{R}$ is the measurement noise variance, and $z_k \in \mathbb{R}$ is the validated position measurement passed by the chi-squared outlier gate. The term $(z_k - \mathbf{H}\hat{\mathbf{x}}_{k|k-1})$ is the innovation which is the signed discrepancy between the observed and predicted position. The updated state estimate $\hat{\mathbf{x}}_{k|k}$ blends the prior prediction with this innovation, weighted by the Kalman gain $\mathbf{K}_k \in \mathbb{R}^{2\times1}$. The filter produces two outputs that feed directly into the control pipeline. The noise-reduced position $\hat{p}_k$ replaces the raw sensor reading as the input to the PID error calculation. The velocity estimate $\hat{v}_k$, derived fully from observing position changes over time, is given to the PID to help anticipate the drone's current motion and issue smoother velocity commands during descent.

3.6 PID Control

The proportional-integral-derivative (PID) controller is a robust feedback control mechanism utilized to drive a system's actual state toward a target state (Set-point) [23]. In the context of 6DoF drone tracking and precision landing, the system requires independent control across 3 spatial dimensions. Consequently, three separate PID controllers are deployed for the X, Y (Altitude), and Z axes [24].

For a specific axis, the controller continuously calculates an error signal, $e(t)$, which represents the difference between the target position and the current sensor reading at time t:

$$e(t) = Target - Current(t) \tag{3.10}$$

The control signal $u(t)$, which corresponds to the velocity commands sent to the flight controller via MAVLink, is the sum of three distinct operational terms:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{3.11}$$

4

Design and Implementation

4.1 System Overview

To understand the various subsystems of this project, there is a visualization of the whole system as shown in Figure 4.1. The core processing takes place on the host computer (LattePanda), which runs the control algorithm for landing.

The position data comes directly from the Sixdof Kipa optical sensor mounted on the UAV, which tracks the ground-based IR Beacons. The LattePanda processes these real-world coordinates through three filters alongside PID control logic and generates velocity commands. These commands are sent via the MAVLink from UDP to the physical serial/UART connection on the Pixhawk Flight Controller to actuate the drone motors.

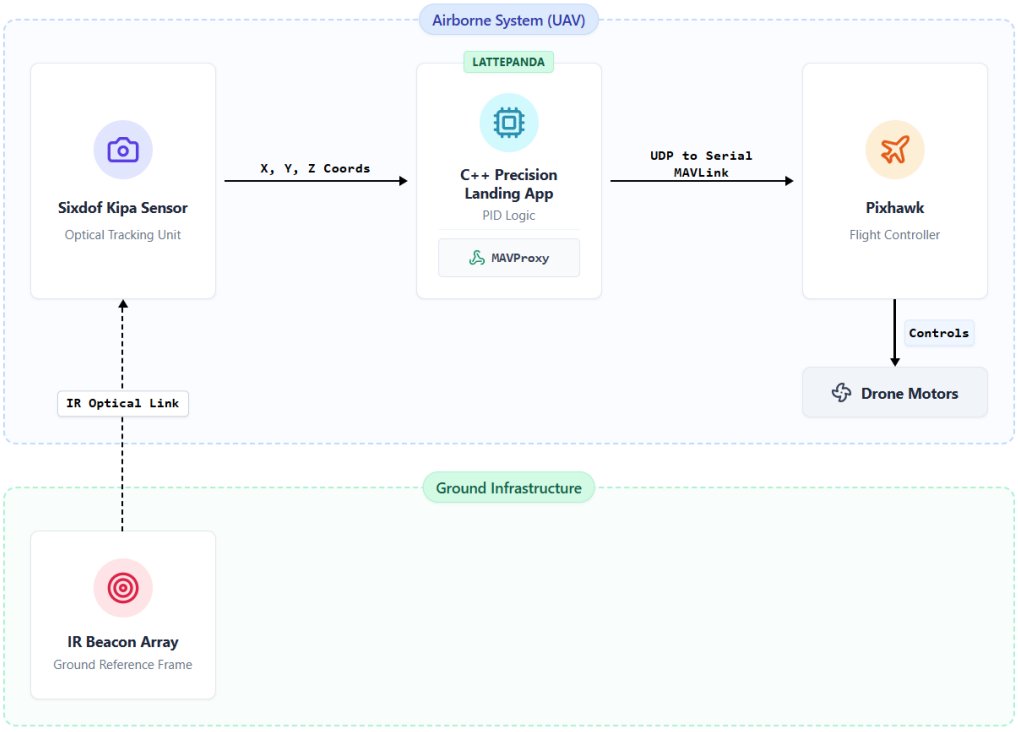

Figure 4.1: System Architecture

4.2 Hardware Integration

The physical integration of this precision landing system relies on a decentralized computing architecture. The LattePanda companion computer acts as the primary processing node, interfacing simultaneously with the Sixdof Kipa optical sensor and the Pixhawk flight controller.

The Kipa sensor is connected to the LattePanda through a Ethernet cable, transmitting high-frequency compressed optical data via UDP packets, and through a USB-MicroUSB cable for power supply. At the same time, the LattePanda is connected to the Pixhawk flight controller through a USB-MicroUSB (for bench test only) or serial interface (TELEM2 for actual flying). Because the control algorithm generates MAVLink velocity commands via a UDP port, a routing utility known as MAVProxy is deployed in the background on LattePanda. MAVProxy acts as a network bridge, capturing the UDP packets generated by the the control algorithm and forwarding them through the physical cable to the Pixhawk flight controller, while simultaneously routing telemetry data back to the ground control station via an Ethernet cable (for bench test only) or a telemetry radio (for actual flying).

During hardware operation, QGroundControl (QGC) runs on a ground operator laptop and connects to the telemetry radio. QGC provides the operator on ground with the Pixhawk's arm status, active flight mode, EKF2 health, and sensor diagnostics. Before each flight the operator need to use QGC to confirm that: (1) The EKF2 has accepted the position data feed from the Kipa sensor. (2) The host computer has successfully commanded `Offboard` mode, visible in QGC's flight mode indicator. (3) The drone is armed, confirmed by the arm status indicator. If these conditions are not met, the operator can abort the mission before the autonomous precision landing begins.

4.2.1 Optical Tracking Configuration

As mentioned in Chapter 2, the proprietary Kipa sensor actively tracks the 6DoF data of the UAV. However, in the configuration of it, it is worth to note that unlike the usual perception of the axes, X and Z constitute the horizontal position data, while Y constitutes the altitude data.

To provide a robust spatial tracking, a constellation of eight IR beacons is geometrically distributed around the corners or boundary of the landing pad. Normally, The effective tracking range is proportional to the extent of separation between the beacons. The longer the tracking distance, the larger separation space is needed which means a larger landing pad is needed. For our tracking range (approximately 20 m), The precise coordinates of the beacons (x, y, z), measured in meters, are configured as Table 4.1.

As illustrated in Table 4.1, the beacons are taped on a landing pad which has a perimeter of 0.5 m $\times$ 0.4 m, on the ground ($Y = 0$). This specific configuration guarantees that the Kipa sensor maintains line-of-sight (LoS) with multiple beacons simultaneously, regardless of the UAV's yaw orientation during the descent.

Table 4.1: Spatial Coordinates of the IR Beacon Constellation

Beacon ID	X (meters)	Y (meters)	Z (meters)
B1	0.25	0.00	0.20
B2	-0.25	0.00	0.20
B3	-0.25	0.00	-0.20
B4	0.25	0.00	-0.20
B5	0.00	0.00	0.20
B6	-0.25	0.00	0.00
B7	0.00	0.00	-0.20
B8	0.25	0.00	0.00

4.2.2 MAVLink Routing Architecture

The Pixhawk 6C connects to the LattePanda via a USB serial cable. Since both the control application and QGroundControl need to communicate with the Pixhawk simultaneously over this single cable, the `mavlink-router` runs as a background daemon on the LattePanda to manage the traffic.
`mavlink-router` sits between three endpoints. It connects to the Pixhawk over the serial port. It forwards telemetry to QGroundControl on the ground laptop via UDP. It also communicates with the control application locally on the LattePanda. For incoming telemetry, `mavlink-router` forwards `HEARTBEAT` messages from the Pixhawk to the control application via UDP. The application reads these to monitor flight mode and arm status.
For outgoing commands, the control application sends computed velocity commands to `mavlink-router` via transmission control protocol (TCP). The `mavlink-router` then forwards them to the Pixhawk over the serial cable. The `TCP_NODELAY` option is set to ensure that each command is sent immediately without buffering.

4.2.3 EKF2 Vision Integration

For the Pixhawk to use the Kipa sensor as its position reference instead of GPS, the filtered position data must be sent to the Pixhawk's EKF2 in a format that it understands. PX4 firmware provides a dedicated MAVLink message for this purpose: The control application sends the `VISION_POSITION_ESTIMATE` message to the Pixhawk every 20 ms, carrying the Kipa's filtered (x, y, z) position. Upon receiving it, the EKF2 treats this as an external vision position source and fuses it with the IMU data to maintain a stable state estimate without GPS.
This replaces GPS as the primary position data source in the EKF2, which simply tells Pixhawk that the autonomous landing does not need any GPS to work. The relevant PX4 parameters are configured via QGC as follows:

- `EKF2_EV_CTRL` = 15 — enables the fusion of vision position data and velocity into the EKF2 state estimate
- `EKF2_HGT_REF` = 3 — sets vision as the primary height reference, replacing the barometer

- `EKF2_EV_DELAY` = 0 — instructs the EKF2 not to apply any time-delay compensation for the incoming vision position data and indicates the delay is negligible.The end-to-end delay from sensor capture to EKF2 receipt is reasonably small so that compensation would not improve state estimation accuracy for a low-speed precision landing approach.

4.2.4 Ground Station Monitoring

During hardware integration and bench testing, QGroundControl runs on a ground operator laptop and connects to the LattePanda via Ethernet (bench) or a telemetry radio (flight). The user needs QGC to verify three conditions before each test:

1. The EKF2 has accepted the Kipa vision position estimate, indicated by `Ready-to-Fly` status.
2. The host computer has successfully commanded `Offboard` mode, in QGC's flight mode indicator.
3. The drone is `Armed`, confirmed by the arm status indicator turning green.

If any condition is not met, the operator aborts before the autonomous landing begins.

4.3 Software Architecture

The precision landing application is developed in C++ to guarantee low-latency execution and efficient memory management.

4.3.1 Dual-Execution Modes

The control application supports two execution configurations in our self-defined mode selection (`#define SIMULATION_MODE`).
In simulation mode, the application connects to Mission Planner's ArduPilot SITL environment via TCP, receiving simulated drone position from the
`LOCAL_POSITION_NED` MAVLink stream. This configuration was used for the systematic robustness testing reported in the result chapter, where configurable wind disturbances parameters were configured into the simulation physics model to evaluate the control algorithm across a wide range of conditions before physical deployment.
In hardware mode, the application subscribes to the Sixdof SDK's `Pose6Dof` callback, receiving live 6DoF position data from the physical Kipa sensor. This configuration runs on the LattePanda with the PX4-powered Pixhawk 6C, using QGC as ground control. The control logic is identical between both modes — only the position data source and communication target differ.

4.3.2 Concurrency Model

To ensure the high-frequency control loop is never blocked by network latency, the application uses three concurrent threads communicating through `std::atomic` shared variables [25]:

- **Sensor thread**: It reads the filter pipeline and PID controllers on each new Kipa frame at up to 400 Hz and writes computed velocity commands to atomic variables.
- **TX thread**: It reads velocity atomics and transmits `SET_POSITION_TARGET_LOCAL_NED` MAVLink messages at a fixed 10 Hz, independent of sensor state. This rate is chosen for two reasons: it satisfies PX4's `Offboard` mode keepalive requirement, which triggers a `Failsafe` if nothing is received within 500 ms, while remaining well within the serial link's 115 200 Bd bandwidth constraint. Transmitting at the full 400 Hz sensor rate would exceed the available serial bandwidth and is unnecessary for the relatively low-speed precision landing approach.
- **RX thread**: It waits for `HEARTBEAT` messages from the Pixhawk, forwarded via `mavlink-router`. The `HEARTBEAT` is a standard MAVLink message that the Pixhawk broadcasts automatically at 1 Hz. It is a health signal that confirms the flight controller is still active [26]. Other than that, it also carries the Pixhawk's current flight mode and arm status. The RX reads these two values and writes them to atomic variables, allowing the state machine (Section 4.3.3) to monitor the Pixhawk's operational state and trigger the appropriate transitions. For example, it confirms that `Offboard` mode has been accepted before it needs to start landing phase.

4.3.3 Finite State Machine

The system's operational behavior is governed by a Finite State Machine (FSM) implemented as an `std::atomic<SystemState>` variable, ensuring safe reads across all three threads. Four states are defined:

- **STANDBY** — The system remains in this state on startup, waiting until three conditions are met: the EKF2 has a valid position estimate, the Pixhawk has accepted `Offboard` mode, and the motors are armed.
- **LANDING** — The active landing state in which the PID controllers issue continuous velocity commands toward the pad center. The system remains in this state as long as the Kipa sensor maintains locking on the landing pad's beacons and the weather severity (in Section 4.4.2) is within acceptable bounds.
- **LOITER** — Entered this state when either the Kipa sensor loses tracking on the beacons or the weather severity reaches `EXTREME`. The descent is halted and all velocity commands are set to zero, causing the drone to hold its current position until the conditions are improved.
- **TOUCHDOWN** — Entered this state when both the horizontal and vertical errors is in the tolerance limit, confirming a successful landing. Velocity commands are zeroed and the system waits for the operator to disarm the motors.

Key state transitions based on the conditions discussed above:

- **STANDBY → LANDING**
- **LANDING → LOITER**
- **LOITER → LANDING**

- **LANDING → TOUCHDOWN**

4.3.4 Automated Arming and Offboard Switching

As described in Section 4.3.3, the system exits `STANDBY` only when EKF2 has position data, `Offboard` mode is confirmed, and the motors are armed. This subsection describes how those three conditions are achieved through our control application, independent of the FSM:

1. **EKF2 stability check** — the RX thread closely monitors the altitude data reported by `LOCAL_POSITION_NED`. Once it reports non-zero values (greater than 5 cm) for at least 3 s, the EKF2 has produced a valid position estimate and the sequence proceeds.
2. **Offboard mode switch** — A mode change command is sent to the Pixhawk 6C via `MAV_CMD_DO_SET_MODE`, requesting it to enter `Offboard` mode. PX4 has a very strict safety requirement for this: it will accept the switch into `Offboard` mode only if velocity setpoint messages are being streamed to it continuously. Without an active stream, PX4 rejects the request to prevent the drone from entering autonomous control without a solid command source. This requirement is satisfied by the TX thread, which transmits velocity commands at 10 Hz continuously once the program starts. When the sensor is tracking, it sends the latest computed velocity commands. When the sensor loses lock, it continues streaming the last known values, keeping the `Offboard` connection alive until the state machine detects the dropout and zeros the commands by transitioning to `LOITER`.
3. **Arm command** — once `Offboard` mode is confirmed by the incoming `HEARTBEAT`, `MAV_CMD_COMPONENT_ARM_DISARM` is sent to arm the motor. If the drone is subsequently disarmed unexpectedly, all flags reset and the sequence restarts automatically.

4.4 Control Algorithm Implementation

4.4.1 Control Objective

The objective of the control algorithm is to guide the UAV to the geometric center of the landing pad. Let the UAV's position at time t be $P(t) = [X(t), Y(t), Z(t)]^T$. The algorithm continuously computes corrective velocity commands to drive the spatial error vector to zero:

$$\lim_{t \to T_{\text{land}}} E(t) = \lim_{t \to T_{\text{land}}} \begin{bmatrix} 0 - X(t) \\ 0 - Y(t) \\ 0 - Z(t) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.1}$$

4.4.2 Signal Processing Pipeline

Raw position data from the Kipa sensor needs to go through three sequential stages before reaching the PID controllers. Each axis (X, Y, Z) is processed independently.

Median Filter
The median filter is instantiated with a sliding window of $N = 5$ samples, rejecting single-frame impulse anomalies. For example, a raindrop that momentarily obscures an IR beacon.

Chi-Squared Outlier Gate
The outlier gate is configured with a 20-sample history window and threshold $\tau = 9.0$ ($\approx 3\sigma$). If a new measurement causes the Mahalanobis distance to exceed τ, it is rejected and the last valid coordinate is held, preventing the PID from reacting to erroneous bursts caused by environmental occlusions.
However, a prior implementation of this gate introduces a deadlock problem. If the drone is genuinely displaced to a new position by a sustained gust, every subsequent reading from that new position will be rejected because the 20-sample history still reflects the old position. The history never updates because only accepted values enter it, causing the gate to permanently report the old position regardless of how long the drone remains at the new one.
To resolve this, a consecutive rejection counter was added. If five consecutive measurements are rejected, the gate concludes the drone has genuinely moved to a new position rather than experiencing a transient spike. It then resets the history window, accepts the new position, and resumes normal operation:

$$\hat{z}_k = \begin{cases} z_k & \text{if } \chi^2 \leq \tau \text{ (accepted)} \\ z_k & \text{if rejection count} \geq 5 \text{ (history reset)} \\ z_{last} & \text{otherwise (rejected)} \end{cases} \tag{4.2}$$

The five-rejection threshold corresponds to approximately 100 ms at the sensor callback rate. A displacement persisting longer than 100 ms is treated as genuine drone movement rather than sensor noise, allowing the gate to track the drone's true position under sustained wind disturbance while still rejecting brief transient spikes.

Kalman Filter and Weather Estimation
The 1D Kalman filter is for mitigating residual Gaussian noise that passes through the previous two stages and produces a velocity estimate used by the PID controller during descent. An important tuning parameter is the measurement noise R, which controls how much can the filter trust the incoming sensor measurements versus its own kinematic prediction. A higher R indicates less trust in the sensor. The baseline values are set to $R = 0.020$ for the horizontal axes (X, Z) and $R = 0.010$ for the vertical axis (Y). This smaller R in Y means the higher importance of altitude accuracy during descent.
To adapt the filter's behavior to changing outdoor conditions, a `WeatherEstimator` module runs alongside the filter and continuously estimates the current level of disturbances in environment. It achieves this by measuring how much the raw sensor readings deviate from the Kalman filter's output. Larger deviations indicate more turbulent or noisy conditions. Specifically, for each sensor callback, the horizontal residuals r_x and r_z which are the differences between the raw chi-squared outlier gate output and the Kalman filter's smoothed estimate on the X and Z axes, are combined together into a single magnitude:

$$m_i = \sqrt{r_{x,i}^2 + r_{z,i}^2} \tag{4.3}$$

The root mean square (RMS) of this magnitude is then computed over a rolling window of the last $N = 40$ samples:

$$s = \sqrt{\frac{1}{N}\sum_{i=1}^{N} m_i^2} \tag{4.4}$$

Based on the s, the environment is divided into four severity levels. This severity level is then used to dynamically adjust both the Kalman filter's measurement noise R and the PID gain scaling factors throughout the descent, as described in Section 4.4.4.

4.4.3 Discrete-Time PID Implementation

Three independent `AdaptivePID` instances control the X, Y, and Z axes. Although digital controllers inherently operate in discrete time, we implement a *dynamic* time step Δt rather than a fixed time step. The Kipa sensor keeps sending position data at up to 400 Hz but with minor variations due to scheduling jitter, described in Section 5.4.2. Using a fixed Δt would introduce systematic errors into the integral accumulation and derivative calculation on every callback. Instead, Δt is measured precisely on each callback using a high-resolution clock, ensuring mathematically correct integration and differentiation regardless of timing variations.
The discrete approximations at the k-th callback are as follows.
Proportional term:

$$P_k = K_p \cdot e_k \tag{4.5}$$

Integral term:

$$\text{IntegralError}_k = \text{IntegralError}_{k-1} + (e_k \cdot \Delta t), \qquad I_k = K_i \cdot \text{IntegralError}_k \tag{4.6}$$

Derivative term:

$$D_k = K_d \cdot \frac{e_k - e_{k-1}}{\Delta t} \tag{4.7}$$

Output:

$$u_k = P_k + I_k + D_k \tag{4.8}$$

The base gains were determined empirically through iterative tests in the Mission Planner's SITL environment. We progressively increase each gain from zero while observing the drone's convergence behavior under simulated wind conditions. The final values are $(K_p, K_i, K_d) = (0.80, 0.15, 0.06)$ for the horizontal axes and $(1.00, 0.10, 0.00)$ for the vertical axis. The vertical axis uses zero derivative gain because altitude control takes precedence. A smooth, steady descent is more important than fast error correction which could lead to erratic throttle commands.
It should be noted that these gain values are specific to the drone simulated. UAVs of different frame sizes, motor configurations, or payload weights will have different responses to the same velocity commands. Generally, a heavier drone accelerates more slowly, while a lighter drone may overshoot more easily. Consequently, the

base gains in here should be treated as a starting point rather than universal practices. Any change in drone or payload variation would require the gain values to be reconsidered and retested before any implementation.

4.4.4 Weather-Adaptive Gain Scheduling

The gain scheduling mechanism depends on a real-time severity classification produced by a `WeatherEstimator` module. The estimator continuously calculates the RMS of the horizontal positional residuals which are the difference between the chi-squared gate's accepted output and the Kalman filter's smoothed prediction in over a rolling window of $N = 40$ samples:

$$s = \sqrt{\frac{1}{N}\sum_{i=1}^{N} m_i^2}, \qquad m_i = \sqrt{r_{x,i}^2 + r_{z,i}^2} \tag{4.9}$$

This residual is large only when the position signal changes faster than the Kalman filter can track, which occurs under turbulent gusts rather than steady wind. Because a constant wind speed of any magnitude produces a smooth, predictable trajectory that the Kalman filter follows closely, keeping the residual near zero. Therefore, the score s is a indicator for turbulence intensity rather than the absolute wind speed.

Based on this score, the environment is divided into four severity levels:

- `CALM` — $s < 0.03\,\text{m}$: signal stable, no disturbance detected
- `MODERATE` — $s < 0.08\,\text{m}$: mild turbulence, slightly degraded signal
- `SEVERE` — $s < 0.18\,\text{m}$: significant turbulence, filter prediction less reliable
- `EXTREME` — $s \geq 0.18\,\text{m}$: heavy turbulence, position signal highly erratic

Because the score is averaged over 40 samples, spikes dilute progressively into them as new calm samples enter the window while the old spike samples age out. Hence, severity levels reflect a short-term history of disturbance rather than instantaneous noise.

At each control cycle, the effective PID gains are scaled by the current severity [27]:

Table 4.2: PID Gain Scaling Factors per Weather Severity Level

Severity	CALM	MODERATE	SEVERE	EXTREME
κ_p	1.00	0.85	0.70	0.55
κ_i	1.00	0.70	0.40	0.15
κ_d	1.00	0.80	0.60	0.40

Under `SEVERE` or `EXTREME` conditions the integral accumulator is nearly frozen in order to prevent windup. That means when the position signal is highly erratic, allowing the integral to continue accumulating would cause it to build commands based on corrupted measurements, which potentially, can drive the drone in the wrong direction once conditions improve. The complete control output at severity level κ is:

$$u_k = \kappa_p K_p\, e_k + \kappa_i K_i\, I_k + \kappa_d K_d \frac{e_k - e_{k-1}}{\Delta t} \tag{4.10}$$

4.4.5 Derivative Filtering and Velocity Feed-Forward

The derivative term of a PID controller reacts to how quickly the error is changing. In theory this helps the controller reduce oscillation. The raw finite-difference approximation is shown as follows:

$$\frac{e_k - e_{k-1}}{\Delta t} \tag{4.11}$$

But it creates a problem in actual practice. The position data from the Kipa sensor, even after our 3 stage filters, still contain small residual noise or fluctuations between consecutive readings. When two noisy readings are subtracted and divided by a very small time step at 400 Hz, those tiny differences become large numbers. The derivative term therefore sees rapid and probably random fluctuations that have nothing to do with actual drone motion, but it commands the drone to correct noise that is too small and should be ignore in practice. This will lead to a drone that twitches and jerks rather than making a smooth descent.
To address this problem, a first-order exponential moving average (EMA) filter is applied in the derivative term before it enters the PID output. Rather than using the raw derivative directly, the filter mixes it with the previous result:

$$d_k^{\text{filt}} = \alpha \, \frac{e_k - e_{k-1}}{\Delta t} + (1 - \alpha) \, d_{k-1}^{\text{filt}} \tag{4.12}$$

Values of $\alpha = 0.20$ for the horizontal axes and $\alpha = 0.40$ for the vertical axis are used in this case. Stronger smoothing is applied to the horizontal axes since horizontally, the X, Z position data tend to carry more noise from environmental disturbances. The vertical axis smoothing is not as great as horizontal axes because altitude data are tend to be more stable. Also, the descent governor uses the horizontal error to decide whether the drone's vertical descent is permitted at all, that means the vertical PID must respond cleanly to altitude changes once descent is allowed.
Besides the filtering, the derivative channel in PID also includes a velocity term. The standard PID only reacts once a position error already exists. But by the time it detects those errors, the drone is moving fast toward the pad, which might be overshot slightly. The Kalman filter provides a real-time velocity estimate v_{KF} that tells the PID how fast the drone is currently moving which provides a timely information for the output result:

$$u_{\text{ff}} = -K_d \cdot v_{\text{KF}} \cdot 0.5 \tag{4.13}$$

The feed-forward term is added directly to the total PID output with the proportional, integral, and the filtered derivative terms. So the complete velocity command for each axis is as follows:

$$u_k = P_k + I_k + D_k^{\text{filt}} + u_{\text{ff}} \tag{4.14}$$

4.4.6 Descent Governor

The descent governor enforces our speed limits on altitude.It is there to make sure the drone achieves a certain level of horizontal alignment to the landing pad's center

before further descent [28]. Three gates are defined as:

Table 4.3: Descent Governor Gate Configuration

Gate	Altitude	Max $\dot{y}$	e_h limit	e_h (SEVERE)
1	> 3 m	0.45 m/s	0.50 m	0.35 m
2	> 1 m	0.22 m/s	0.30 m	0.20 m
3	≤ 1 m	0.10 m/s	0.15 m	0.10 m

The maximum descent rate is further multiplied by a weather severity factor $f \in \{1.0, 0.80, 0.60, 0.40\}$ for CALM through EXTREME. If weather reaches `EXTREME` below 5 m, descent is suspended entirely and the FSM transitions to `LOITER`.

4.4.7 Velocity Clamping and Cross-Axis Descent Hold

A hard saturation of 0.5 m/s is applied to all velocity commands:

$$V_{\text{cmd}} = \max(-V_{\max},\, \min(V_{\max},\, u_k)) \tag{4.15}$$

Horizontal velocity is additionally bounded by an altitude-proportional limit to ensure gentle corrections near the ground:

$$v_{h,\max} = \text{clamp}(0.8\,|y|,\ 0.15,\ 1.0)\ \text{m/s} \tag{4.16}$$

To prevent descent before horizontal alignment is achieved, the vertical velocity command V_Y is overridden to zero whenever the horizontal Euclidean error exceeds a threshold:

$$E_{\text{horizontal}} = \sqrt{(0 - X)^2 + (0 - Z)^2} > 0.20\,\text{m/s} \ \Rightarrow\ V_Y = 0 \tag{4.17}$$

This compels the drone to hover and correct horizontal position before resuming descent, which significantly reduces the risk of a misaligned touchdown.

5
Results

This chapter presents results from three distinct evaluation streams: software-in-the-loop (SITL) simulation experiments validating the control algorithm, physical characterization of the Kipa optical sensor's range performance, and a computational latency measurement of the full sensor-to-command pipeline.

5.1 SITL Simulation Results

5.1.1 Test Methodology

All simulation experiments were in Mission Planner with the ArduPilot environment. This simulation replaces the Kipa sensor data with the data in the SITL environment including position, velocity, attitude. The reason the Kipa sensor cannot be used in this simulation is because the drone's position are in the simulated environment and cannot be fed in with the real world data.

The sensors in the simulation constitute the three kinds of data above. Since there is no Kipa sensor input, the GPS, barometer data have to be used in this case to imitate the feedback of the Kipa sensor. This simulation is only for testing the functional correctness and a certain degree of robustness of our control algorithm. Therefore, it does not contradict with our GPS-denied environment landing because in real-world implementation of the control algorithm, the input data will be replaced by the Kipa sensor data.

We applied a range of sensor noises in the simulation environment including IMU accelerometer noise ($\sigma = 0.3\,\mathrm{m/s^2}$), gyroscope noise ($\sigma = 0.05\,\mathrm{rad/s}$), GPS position noise (1.0 m), and barometer altitude noise (0.5 m). The frame type was set to X-configuration quadrotor UAV (`FRAME_TYPE = 1`). All of the landings are from 5 m height, either spawning at North, East or North-East (all about 30 m away from the landing origin).

It is also critical to note that even if all of these noises are implemented in the simulation sensors, there is still a huge gap between the simulation fidelity and the real-world condition. For example, the environment of real world is constantly changing (light condition, wind speed and direction), the weight distribution of each drone are different, and each drone's maintenance conditions are different, Kipa sensor cannot have the exact data of the drone as in the simulated case (even though noises are added, the data in simulation case is still far more precise), the battery's remaining capacity can also affect the drone's motors, etc.

Hence, it is important to stress that the SITL results should not be used for com-

parison with the actual landing mission's precision, but only for the proof that our control algorithm, mathematically, maintains a certain level of convergence in different wind and turbulence condition and provides the basis for future implementation on the real landing mission.
All results are evaluated in these two thresholds based on our 40 cm × 50 cm landing pad:

- **Pass** — horizontal landing error < 500 mm
- **Excellent** — horizontal landing error < 300 mm

The horizontal error is the Euclidean distance in the XZ plane: $E_h = \sqrt{X^2 + Z^2}$, measured at the moment the drone reaches the set altitude of landing mission, regardless of lateral position.

5.1.2 Phase 1: Baseline Accuracy

Phase 1 is our fundamental precision under very ideal conditions with no wind, no turbulence. It is our baseline accuracy of the control algorithm. The drone approach to the landing origin from 3 different directions which are north, east and north-east. Every direction was tested with 3 runs.

Table 5.1: Phase 1 — Baseline results (no wind, no turbulence, 3 runs per approach direction).

Test	Spawn	Horizontal Error (mm)			Mean (mm)	Result
		R1	R2	R3		
P1-N	North	0.4	4.8	22.3	9.2	Excellent
P1-E	East	18.4	24.2	4.7	15.8	Excellent
P1-NE	Diagonal	13.9	10.9	12.7	12.5	Excellent

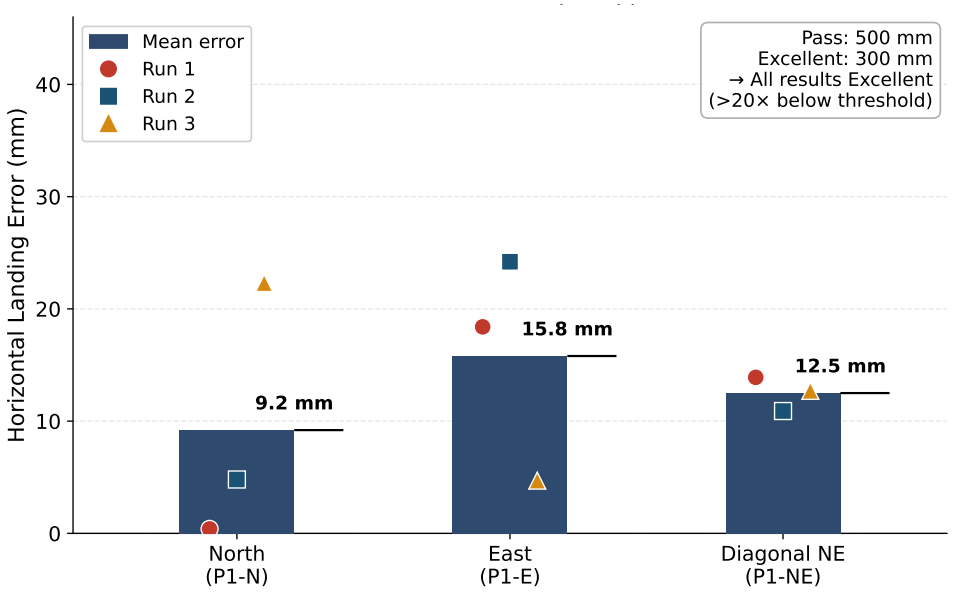

Figure 5.1: Phase 1 baseline landing errors

All 9 runs in this ideal configuration achieved excellent result. Mean errors are ranged from 9.2 mm to 15.8 mm. This proves the theoretical correctness of our control algorithm which reveals that mathematically, our algorithm converges to the designated origin point.

5.1.3 Phase 2: Horizontal Wind Test

In Phase 2, we apply a horizontal wind with different speed to our landing missions. All runs use the diagonal north-east spawn with East wind direction (DIR_Z = 0°, which is a simulation parameter of the vertical angle of the wind direction. DIR_Z = 0° meaning no vertical component of wind direction) and no turbulence.

Table 5.2: Phase 2 — Horizontal error across wind speeds (NE spawn, East wind, no turbulence, DIR_Z = 0°).

Test	Wind (m/s)	Horizontal Error (mm)			Mean	Std	Result
		R1	R2	R3	(mm)	(mm)	
P2-A	2	0.6	20.1	9.8	10.2	8.0	Excellent
P2-B	5	26.9	29.4	3.7	20.0	11.6	Excellent
P2-C	8	21.4	22.8	15.7	20.0	3.1	Excellent
P2-D	10	32.3	23.1	19.3	24.9	5.5	Excellent
P2-E	13	16.7	34.7	15.9	22.4	8.7	Excellent
P2-F	15	23.0	24.1	33.6	26.9	4.8	Excellent

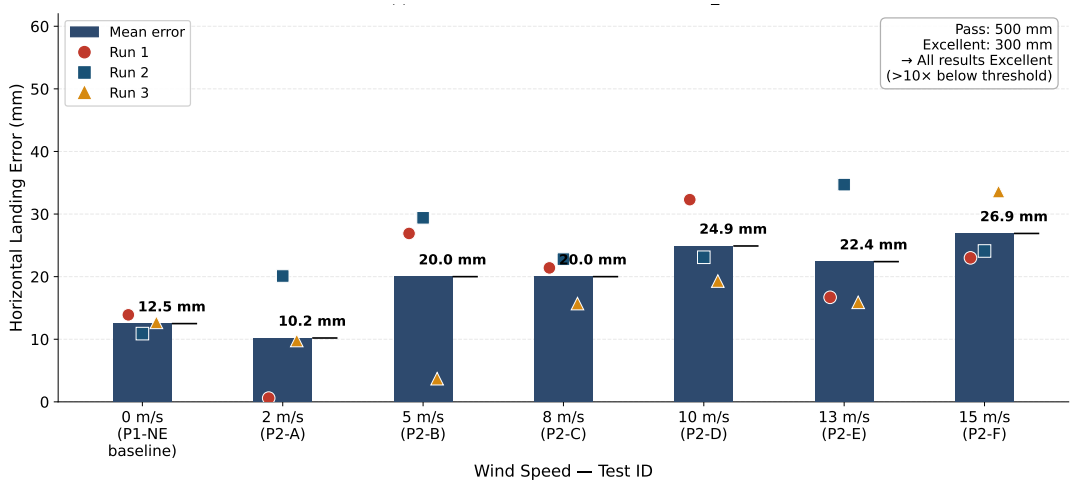

Figure 5.2: Landing error across wind speeds from 0 m/s to 15 m/s

All Phase 2 tests achieved the *Excellent* result. Mean error increased from 10.2 mm at 2 m/s to 26.9 mm at 15 m/s wind speed. This is still far under our *Excellent* threshold. The recorded landing times were stable at approximately 35.6 s across all configurations. A single-direction steady wind is essentially a constant offset force pushing the drone continuously at a fixed magnitude. This can be easily

solved by the integral term of the PID control because over some control cycles, the integral accumulates the persistent error and builds a counter-command to precisely eliminate the disturbance. Therefore, even at 15 m/s wind speed, which is certainly a very extreme condition in the real world, the algorithm is still able to solve it mathematically in simulation. It is critical to warn that in a real environment, the specific drone can vary in its ability to withstand 15 m/s wind speed, especially with additional disturbances. Therefore, this result cannot serve as proof of a real landing in 15 m/s wind, but only as validation for the functional correctness of the PID algorithm. It is not recommended to fly drones in extreme wind conditions.

5.1.4 Phase 3: Turbulence Test

Phase 3 tests slightly more closely resemble real wind conditions but are still far from real. The wind speed parameter is fixed at 10 m/s (East, DIR_Z = 0°, no vertical component) while turbulence intensity is increased from 1 m/s to 9 m/s. At each level, random wind velocity turbulence with a mean deviation of 1 m/s to 9 m/s is imposed on the base wind speed.

Table 5.3: Phase 3 — Landing error across turbulence intensities (NE spawn, 10 m/s East wind, DIR_Z = 0°).

Test	**Turb**	**Horizontal Error (mm)**			**Mean**	**Std**	**Time (s)**	**Result**
		R1	R2	R3	(mm)	(mm)		
P3-A	1	46.7	42.1	61.0	49.9	8.0	35.7	Excellent
P3-B	3	144.8	188.0	151.3	161.4	19.0	35.7	Excellent
P3-C	5	137.1	54.3	122.1	104.5	36.0	37.1	Excellent
P3-D	7	79.9	351.0	240.8	223.9	111.3	38.6	Excellent
P3-E	9	592.2	1263.8	230.7	695.6	428.0	46.1	**Fail**

The system maintained the *Excellent* classification through turbulence 7, with mean errors rising from 49.9 mm to 223.9 mm. The high deviation at turbulence 7 m/s (111.3 mm) reveals the stochastic nature of this disturbance, with Run 1 recording 79.9 mm while Run 2 recorded 351.0 mm under identical nominal conditions. Therefore, each test may yield results of varying accuracy since the wind speed is randomly distributed within the turbulence range.
The large XZ errors observed at turbulence intensity 9 arise from a combination of limitations in our algorithm. The TX thread dispatches velocity commands at 10 Hz, which means up to 100 ms may elapse between successive updates. At $\sigma \approx 9$ m/s, the drone can be physically displaced very far within a single command interval, causing each correction to become obsolete before it reaches the drone. Simultaneously, the integral term accumulates error in one direction across multiple cycles. When the turbulent wind reverses its course, the integral opposes the necessary corrective command until it has fully drained and reversed. This is a process that takes several further cycles. Moreover, the weather condition and cross-axis descent hold thresholds are met at certain moments as gusts momentarily reduce and the

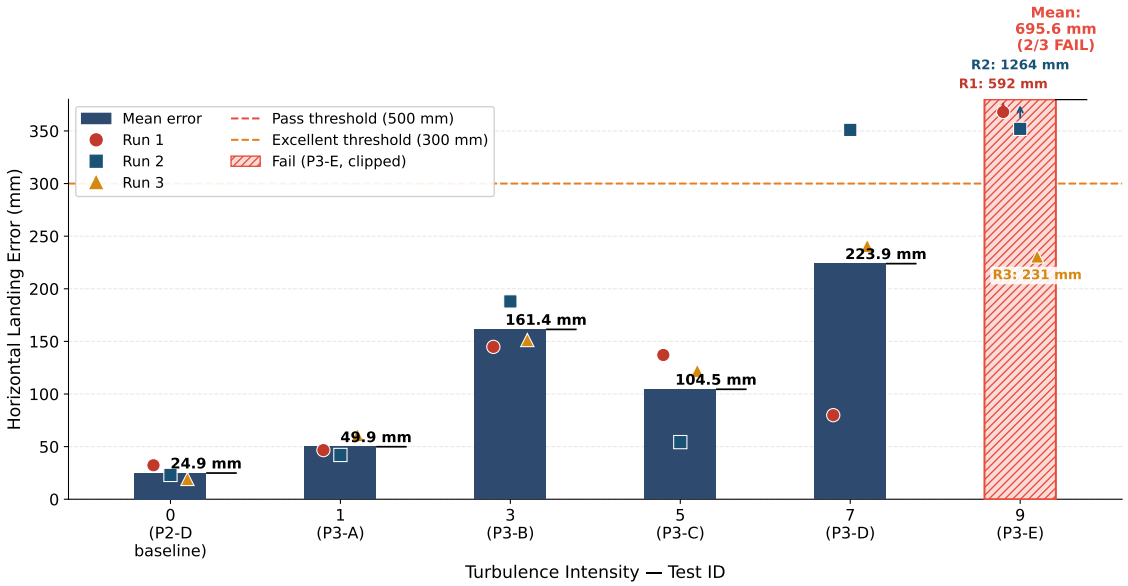

Figure 5.3: Landing error across turbulence intensity levels from 0 (P2-D baseline) to 9.

drone aligns to the XZ origin, permitting a period of time for descending. However, when the Y (altitude) is about to touch the landing altitude, the drone can still be blown significantly off course due to the high turbulence. After reaching the landing altitude, there is no more room to maneuver the XY position because the drone is considered landed, and there is no regaining of height in our algorithm to create more space for realignment in the horizontal position. Together, these effects produce the huge landing errors in this test.

5.1.5 Phase 4: Vertical Wind Test

Phase 4 examines the algorithm under more complex wind condition by adding vertical component of the wind vector. With $windspeed = 10\,\text{m/s}$ and $DIR\_Z = 90°$ (East Wind) fixed, the parameter DIR_Z was used to defined the angles of the wind from $-45°$ (downdraft) through $0°$ to $+90°$ (pure updraft). The wind vector decomposes as:

$$v_h = v\cos(windspeeed), \qquad v_v = v\sin(DIR\_Z) \tag{5.1}$$

All nine configurations achieved the Excellent classification, including the pure updraft case (P4-I) and both downdraft extremes (P4-A). Mean errors ranged from 15.1 mm to 58.3 mm with no systematic trend correlated with elevation angle.

This result, combined with Phase 2 results, reveals the coherence of our algorithm's ability to counteract the wind from different direction. The vertical component of the wind had no observable effect on our landing process which is treated merely as a constant vertical components added to form wind coming from any direction. The control algorithm can handle it well and clean mathematically.

Table 5.4: Phase 4 — Landing error across vertical wind elevation angles (NE spawn, 10 m/s, TURB = 1).

Test	DIR_Z	Vert.	Horizontal Error (mm)			Mean (mm)	Result
	(deg)	(m/s)	R1	R2	R3		
P4-A	−45	−7.1	23.1	54.6	28.2	35.3	Excellent
P4-B	−30	−5.0	39.9	36.7	34.0	36.9	Excellent
P4-C	−15	−2.6	60.1	47.1	67.8	58.3	Excellent
P4-D	0	0.0	58.7	11.7	38.5	36.3	Excellent
P4-E	+15	+2.6	34.8	60.3	65.3	53.5	Excellent
P4-F	+30	+5.0	21.6	41.3	27.9	30.3	Excellent
P4-G	+45	+7.1	5.5	8.4	31.4	15.1	Excellent
P4-H	+60	+8.7	14.1	26.6	21.6	20.8	Excellent
P4-I	+90	+10.0	37.6	26.6	45.1	36.4	Excellent

5.1.6 Phase 5: Combined Worst-Case Stress Test

Phase 5 applies the maximum combined disturbance: 15 m/s wind at DIR_Z = 30° (yielding 13.0 m/s horizontal and 5.0 m/s vertical components) with turbulence intensity 5. Five runs were conducted to obtain a statistical distribution of worst-case performance.

Table 5.5: Phase 5 — Combined worst-case results (NE spawn, 15 m/s, DIR_Z = +30°, TURB = 5, 5 runs).

Run	Horizontal Error (mm)	Time (s)	Result
P5-1	153.4	41.7	Excellent
P5-2	37.2	38.2	Excellent
P5-3	166.7	35.6	Excellent
P5-4	361.9	39.2	Pass
P5-5	187.6	39.0	Excellent
Mean ± Std	**181.4 ± 104.3**	**38.7**	**4/5 Excellent**

All five runs successfully confirmed touchdown within the pass threshold. The mean error of 181.4 mm with a standard deviation of 104.3 mm reflects the stochastic nature of combined turbulence and high wind: Run 2 recorded only 37.2 mm while Run 4 recorded 361.9 mm under nominally identical conditions. This variance is intrinsic to the turbulence model rather than algorithmic instability. The mean error represents a 14.5× increase over the no-wind baseline (12.5 mm) but remains well within the pass threshold. This reflects a high theoretical feasibility of operating in demanding conditions. However, it should be noted again that these results only represent a probabilistic likelihood of achieving a desirable outcome when actually implementing the algorithm. They should not be directly compared with real-world

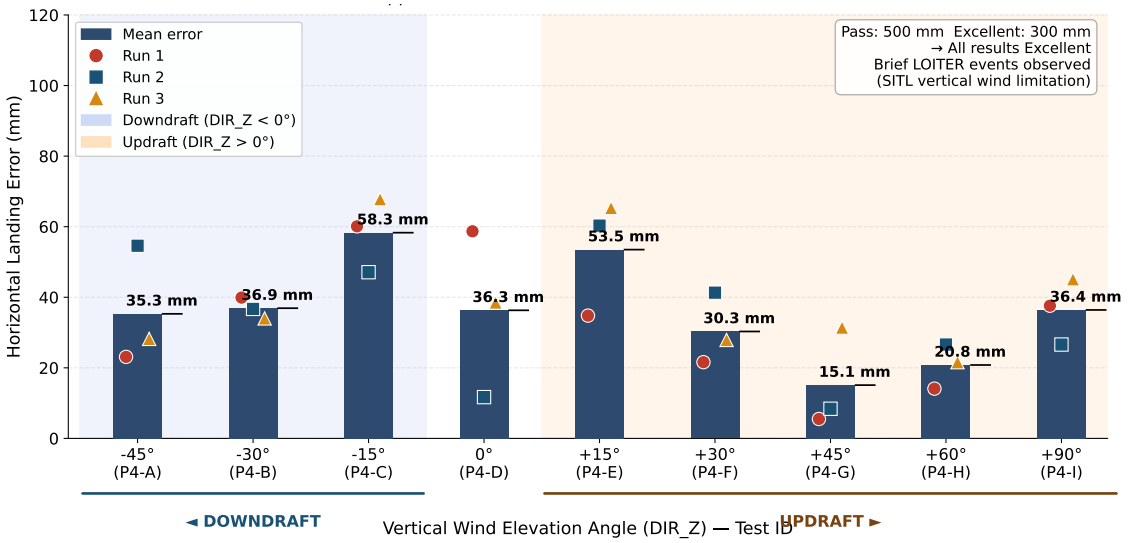

Figure 5.4: Landing error across vertical wind elevation angles.

results, as the simulation conditions remain significantly more ideal than an actual field environment.

5.1.7 Overall Performance Summary

The simulation results reveal two key findings. First, pure horizontal or vertical wind speed has a minor effect on our landing accuracy. The mean error increases by only 2.6× across the full 0 m/s to 15 m/s range, which validates the effectiveness of our algorithm to counter constant disturbances. Second, turbulence is the dominant limiting factor in our tests. Errors grow as turbulence intensity increases, and the results start to degrade severely at intensity 7 and above, with the failure boundary identified at intensity 9.

5.2 Sensor Range Characterization

5.2.1 Test Setup

After validating the control algorithm through simulation, this section reports the physical characteristics of the Kipa optical sensor hardware. The experiments were conducted under standard indoor household lighting conditions. Unlike the simulation tests where position feedback was provided by the SITL simulator, this test used the actual Kipa sensor and an infrared beacon landing pad to directly measure the sensor's tracking performance at different distances.

The landing pad — a rectangular board measuring 50 cm × 40 cm — was placed vertically against a wall with its base resting on the floor as in Fig. 5.6. Eight IR beacons are distributed across the pad surface: four at the corners, and one at the midpoint of each edge, as defined in the beacon map (Table 5.6). The coordinate origin of the beacon map is at the geometric center of the pad, with the Y axis

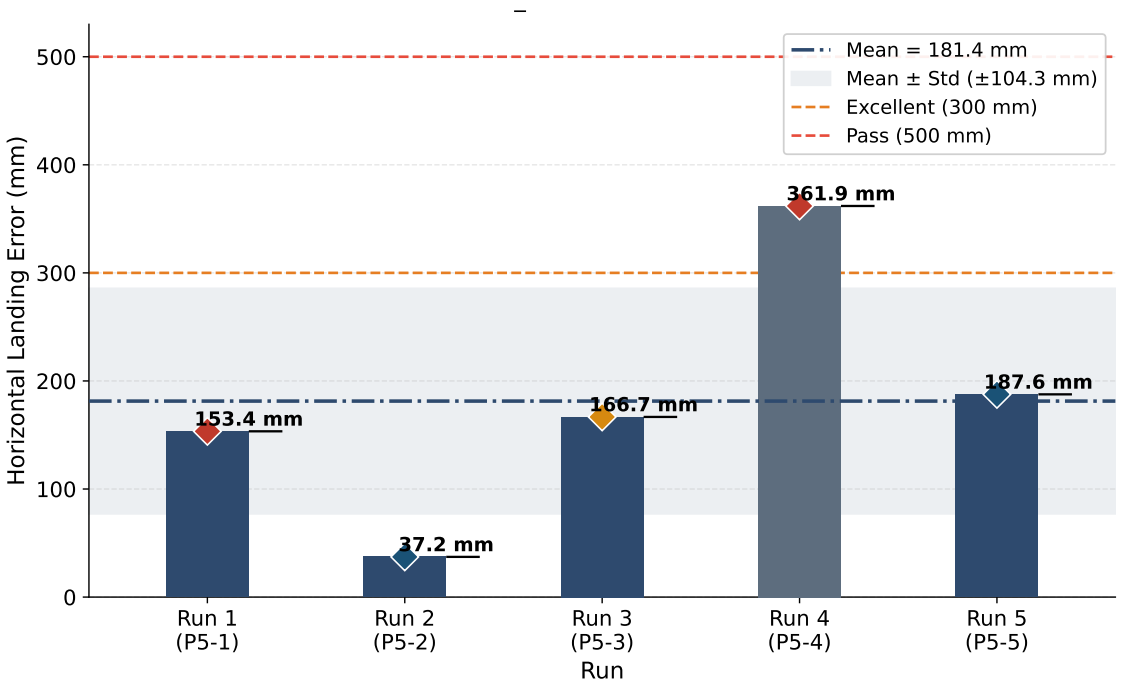

Figure 5.5: Phase 5 combined worst-case results.

pointing outward (perpendicular to the pad surface), X pointing laterally, and Z pointing vertically within the pad plane.

Table 5.6: IR beacon map spatial coordinates.

Beacon ID	Position	X (m)	Y (m)	Z (m)	Description
0	Corner	0.25	0	0.20	Top-right
1	Corner	-0.25	0	0.20	Top-left
2	Corner	-0.25	0	-0.20	Bottom-left
3	Corner	0.25	0	-0.20	Bottom-right
4	Midpoint	0.00	0	0.20	Top-centre
5	Midpoint	-0.25	0	0.00	Left-centre
6	Midpoint	0.00	0	-0.20	Bottom-centre
7	Midpoint	0.25	0	0.00	Right-centre

The Kipa sensor (housing dimensions: 98 mm × 44 mm × 21 mm) was placed on the ground, facing the landing pad, with one side against the adjacent wall. This corner position places the sensor's optical center approximately at ($X \approx 0.25\,\text{m}$, $Z \approx -0.20\,\text{m}$) relative to the landing pad's origin—aligned with the beacon at the lower right corner. Since the sensor is a physical object rather than a point, there is a small inherent offset between the sensor's edge and its optical center, which affects the X and Z measurements. Therefore, a certain deviation from the exact corner beacon coordinates is expected.

To evaluate the sensor's performance over varying ranges, the physical distance between the sensor face and the landing pad surface was measured with a tape

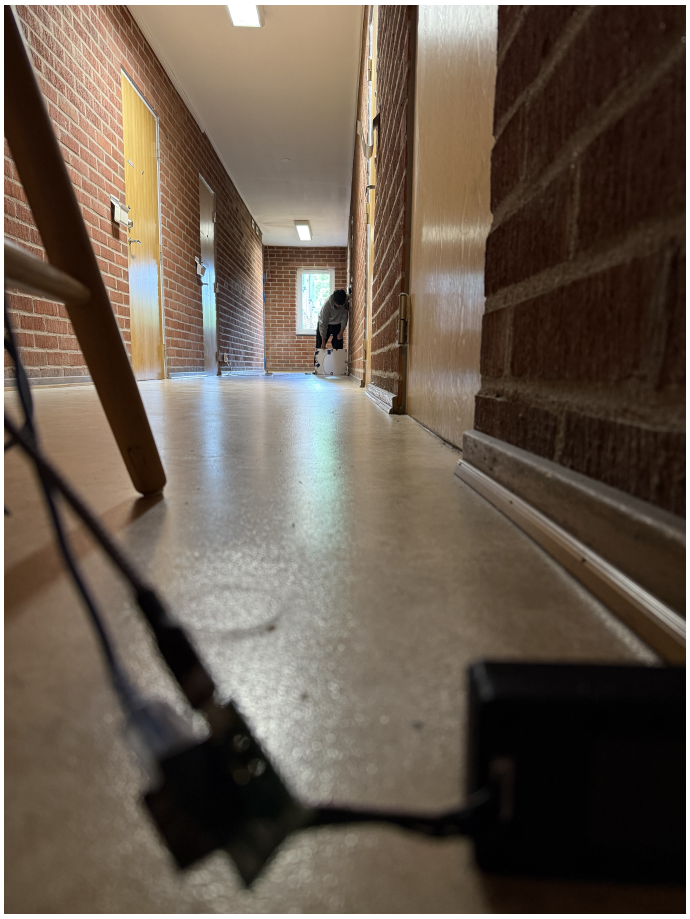

Figure 5.6: Sensor Range Test Setup

measure. The sensor was gradually moved backward in steps to obtain readings at 0.5 m, 1 m, 2 m, 3 m, 5 m, 10 m and 12 m. Five consecutive position readings were recorded at each distance to account for measurement variance.

A separate range ceiling test was conducted outdoors on an overcast day to determine the maximum reliable tracking distance. In this test, tracking was maintained at 10 m, 15 m, 20 m and 23 m (the maximum range of the IR beacon hardware), with only Y-axis depth readings recorded.

5.2.2 Results

Table 5.7 reports the average position and Y depth error reported by the sensor at each test distance. The **Y depth error** is defined as $|Y_{\text{reported}} + d_{\text{physical}}|$, where d_{physical} is the distance measured with a tape measure. The position standard deviation is close to zero (< 0.003 m) across all valid distances, indicating extremely stable tracking when lock is maintained. Since tracking loss occurred, the 13 m row was excluded from the accuracy analysis.

Table 5.7: Indoor sensor range characterization results.

Dist. (m)	X (m)	Y (m)	Z (m)	Y Std Dev (m)	Y Depth Error (m)	Lock	Class.
0.5	0.189	−0.527	−0.230	0.0000	0.027	Yes	**Excellent**
1.0	0.185	−1.028	−0.384	0.0000	0.028	Yes	**Excellent**
2.0	0.178	−2.118	−0.316	0.0005	0.118	Yes	**Acceptable**
3.0	−0.133	−3.267	−0.196	0.0013	0.267	Yes	**Marginal**
5.0	−0.674	−5.271	+0.632	0.0013	0.271	Yes	**Marginal**
10.0	0.353	−11.142	−0.183	0.0033	1.142	Yes	**Fail**
12.0	−1.466	−12.644	−0.158	0.0010	0.644	Yes	**Fail**
13						**No**	**Fail (lost)**

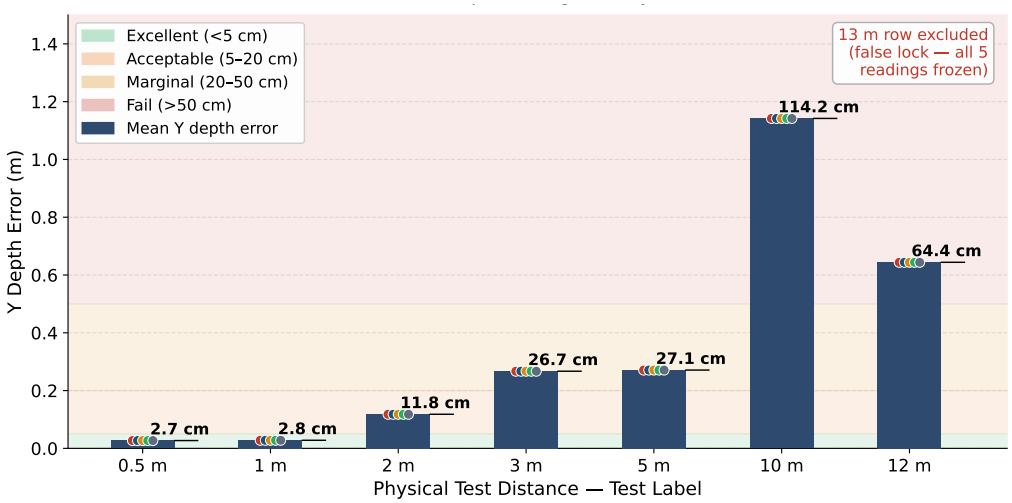

Figure 5.7: Y-axis depth error versus physical test distance.

5.2.3 Discussion

Depth accuracy at short range At 0.5 m and 1.0 m, the depth errors in the Y direction are 2.7 cm and 2.8 cm respectively. This consistent offset at this height is most likely due to a fixed difference between the tape measure reference point (the mat surface) and the internal depth reference of the sensor (its optical center or casing surface). After removing this systematic offset, the net accuracy between 0.5 m to 1 m is less than 1 mm, fully meeting the manufacturer's specification of centimeter-level positional accuracy. This range is the most critical safety zone for the landing system: the final 0.5 m of descent is the moment to make landing decisions, and a depth accuracy of less than 3 cm is already sufficient.

Depth accuracy at mid range At 2 m to 5 m, the depth error in the Y direction increased from 11.8 cm to 27.1 cm. This gradually increasing deviation is consistent with a small angular misalignment between the sensor's movement direction and the

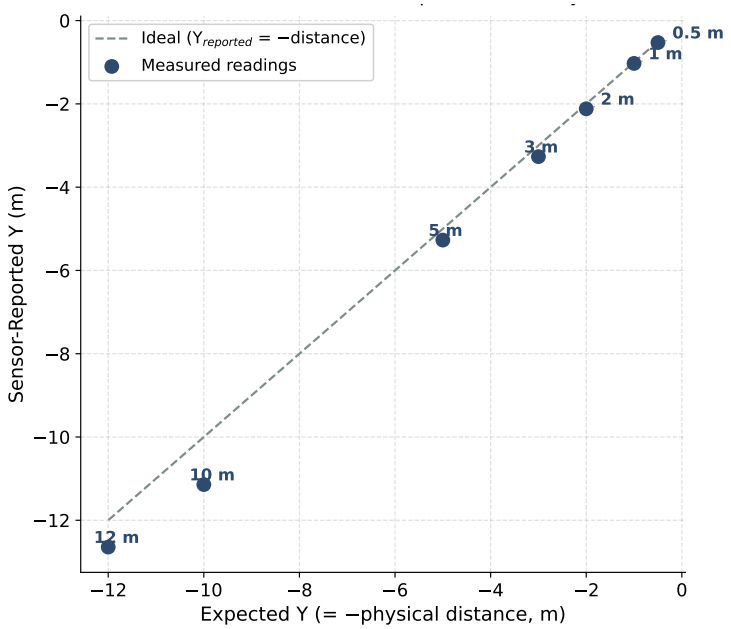

Figure 5.8: Sensor-reported depth (Y) versus actual physical distance.

normal axis of the landing pad. If the sensor is not perfectly perpendicular to the pad when repositioned at each distance, the distance measured with tape will differ from the Y-direction depth reported by the sensor. An angular offset of $\alpha \approx 10°$–$15°$ between the movement path and the Y-axis could cause an error of this magnitude at 5 m. This is a methodological limitation of the manual test setup, rather than a fundamental issue with the sensor's accuracy.

Depth accuracy at long range While these values are relatively large, the positional *stability* (Y-axis standard deviation < 0.003 m) remains excellent at both distances, indicating that the sensor reports consistent positions—the difference lies in the absolute distance rather than tracking noise. The indoor and outdoor readings at 10 m are almost identical (Y-axis readings of -11.14 m and -11.13 m, respectively), confirming that this is due to system measurement bias rather than sensor performance degradation.

Tracking loss at 13 m indoors Indoor tracking was lost at approximately 13 m, compared to approximately 23 m (Table 5.8) in outdoor cloudy weather tests—a 43 % reduction. This difference could be attributed to two factors. First, indoor surfaces (walls, ceilings, floors) reflect infrared pulses emitted by the beacon, creating multipath signals over mid and long distances. This disrupts the encoded infrared detection algorithm due to the low signal-to-noise ratio of the direct path. Outdoor open environments have far fewer reflective surfaces since we conducted the test by elevating the landing pad on a table with no walls or floors immediately around it. Second, standard household lighting emits light closer to the near-infrared spectrum, increasing the sensor's background noise level. The 13 m data supports the false-lock hypothesis: all five readings are fixed at the same value ($X = -3.51$ m, $Z = -5.41$ m), consistent with the tracker locking onto multipath reflections rather than

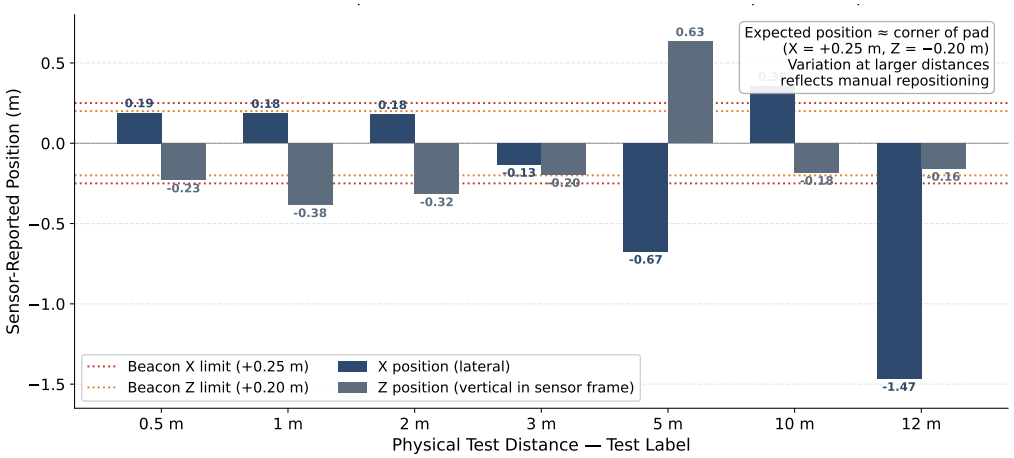

Figure 5.9: Sensor-reported lateral (X) and vertical (Z) positions across distances.

the actual beacon location.

Table 5.8: Outdoor Tracking Y

Y (m)	Y Test(m)
10	-11.125
15	-15.979
20	-20.660
23	-23.219

X and Z position variation The reported X and Z positions vary with distance (Figure 5.9), especially at 3 m, 5 m and 12 m. This is expected: the sensor was manually repositioned at each test distance, and without precise linear guides, slight lateral shifts are unavoidable. The 0.5 m reading ($X = 0.189$ m, $Z = -0.230$ m) most accurately reflects the actual corner position of the sensor, very close to the expected position based on the sensor's physical dimensions ($X \approx 0.25$ m, $Z \approx -0.20$ m).

Implication for the landing system Ranging characteristics confirm that the Kipa sensor provides reliable and stable tracking within the critical landing approach envelope (0.5 m to 5 m) used in the control algorithm. Tracking stability (standard deviation < 0.003 m for all effective distances) indicates that position noise is negligible once target lock is achieved. The 13 m indoor height limit does not affect landing applications—the UAV's final approach altitude is well below this limit. For outdoor deployment, a full 23 m range is available, sufficient to cover any practical initial lock-on altitude.

5.3 Water Protection Sensor Reading Test

Because the IR beacons are not waterproof, it is necessary to cover the beacons to suit the need of landing in rainy environment. To minimize the impact the

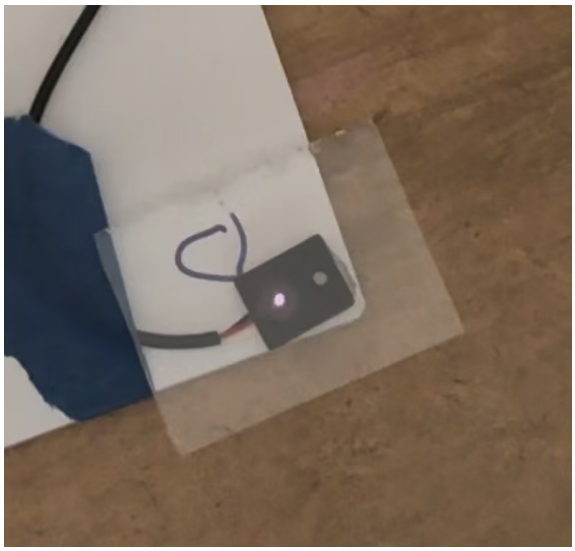

Figure 5.10: Beacon Covered with Acrylic

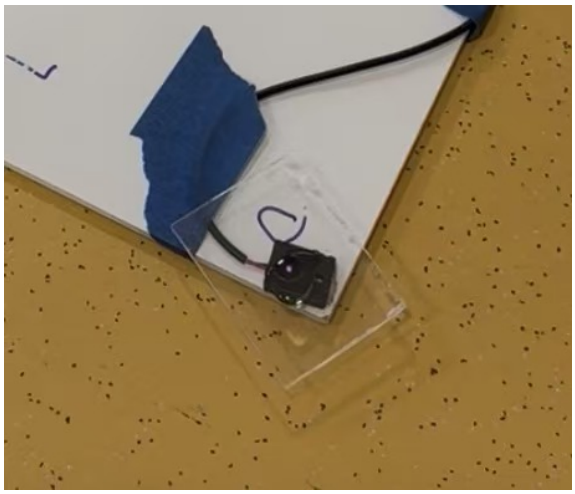

Figure 5.11: Beacon Covered with Acrylic and Droplets

cover does to the IR beacons, we choose transparent acrylics to cover the beacons. Specifically, we cover every beacons with a square transparent acrylic board which is large enough to cover the individual beacon.

Two kinds of test are arranged. The first kind is placing acrylic board directly on the beacons to observe the changes in the reading (see Fig. 5.10). Another kind is water droplets test (see Fig. 5.11), meaning not only did we place the acrylic board on top of the beacons, we also put water droplets on top of the acrylic board to see if the water on top (which is common during the rainy days, water aggregates on the transparent acrylic) can severely affect the sensor readings.

Table 5.9: Test 1: no acrylic

Sample	X (m)	Y (m)	Z (m)
1	-0.031	-0.864	0.071
2	-0.030	-0.864	0.071
3	-0.030	-0.864	0.071
4	-0.031	-0.864	0.071
5	-0.030	-0.864	0.071

Table 5.10: Test 1: with acrylic

Sample	X (m)	Y (m)	Z (m)
1	-0.043	-0.885	0.090
2	-0.043	-0.885	0.090
3	-0.043	-0.885	0.090
4	-0.043	-0.885	0.090
5	-0.043	-0.885	0.090

The Test 1 results are shown in Table 5.9 and Table 5.10. These 2 tables indicate there is no significant drift in sensor readings when we put transparent acrylic board on top of the beacons. Some minor drift can be due to the drift of Kipa sensor, when we holding it during the test. That proves a waterproof trasnparent box is not going to affect the readings of the Kipa sensor.

Table 5.11: Test 2: no acrylic and droplets

Sample	X (m)	Y (m)	Z (m)
1	0.370	-0.881	0.342
2	0.370	-0.881	0.342
3	0.380	-0.882	0.342
4	0.380	-0.882	0.342
5	0.380	-0.882	0.342

Table 5.12: Test 2: with acrylic and droplets

Sample	X (m)	Y (m)	Z (m)
1	0.396	-0.944	0.321
2	0.396	-0.944	0.321
3	0.396	-0.944	0.321
4	0.396	-0.944	0.321
5	0.396	-0.944	0.321

The Test 2 results are shown in Table 5.11 and Table 5.12. In this water droplets test, we can also infer that the droplets on the acrylic pose no significant threats on our sensor readings. Even in rainy days water aggregate on the acrylic waterproof shell, our IR beacons and Kipa sensor can still work together properly.

5.4 Pipeline Latency Characterization

5.4.1 Test Setup

To measure the computational performance of the full sensor-to-command pipeline running on the LattePanda, we conducted a latency characteristic test. This test used a physical Kipa sensor tracking beacon platform in the same indoor environment as the ranging test. No flight controller or MAVLink connection was required. Only the performance of the onboard computation phase was measured.

The pipeline was instrumented with high-resolution timestamps at three points within each sensor callback, using `std::chrono::steady_clock` with nanosecond resolution:

- t_0 — callback entry (sensor data received from SDK)
- t_1 — after filter chain (median pre-filter, chi-squared outlier gate, 1-D Kalman filter, applied to all three axes)
- t_2 — after PID and descent governor (three-axis adaptive PID calculation plus descent rate limiting)

The test ran for 30 s, collecting 11 416 samples at the sensor's native update rate. The TX thread scheduling latency, the additional latency between writing a speed command to a shared atomic variable and the actual dispatch of that command by the TX thread via MAVLink, was not directly measured but was characterized based on the known TX thread cycle of 100 ms (10 Hz).

5.4.2 Results

Table 5.13 summarizes the measured latency at each pipeline stage.

Table 5.13: End-to-end pipeline latency measurements.

Stage	**Mean** (ms)	**Std** (ms)	**Min** (ms)	**Max** (ms)	**p99** (ms)	**Source**
Sensor capture	2.000	—	—	—	—	Manufacturer spec
Callback interval	2.631	0.112	1.171	3.338	2.973	Measured
Filter chain	0.018	0.006	0.004	0.081	0.031	Measured
PID + governor	0.001	0.001	0.000	0.026	0.002	Measured
Total in-callback	0.019	0.006	0.005	0.082	0.034	Measured
TX scheduling lag	50.000	28.868	0.000	100.000	99.000	Analytical (10 Hz TX)
Total E2E (est.)	**52.02**	—	—	—	—	Estimated

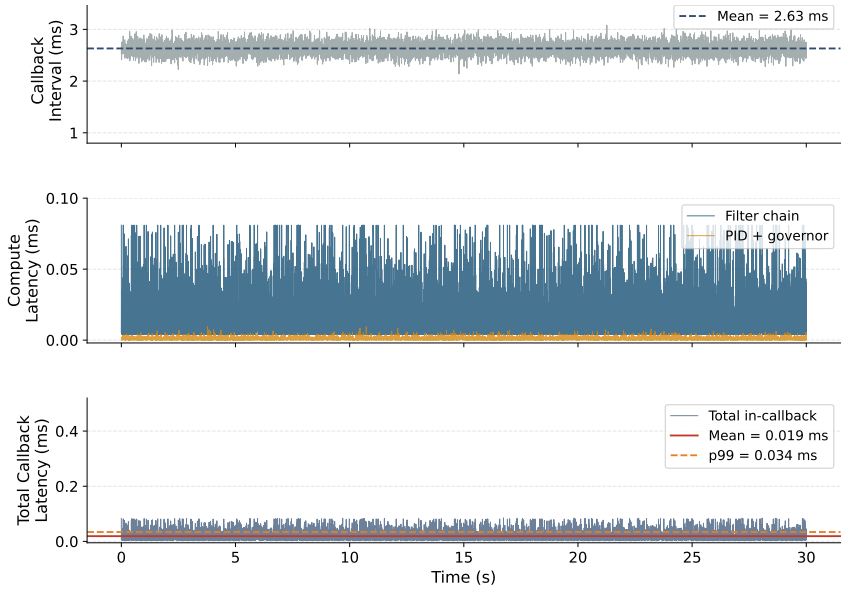

Figure 5.12: 30-second time-series of pipeline latency

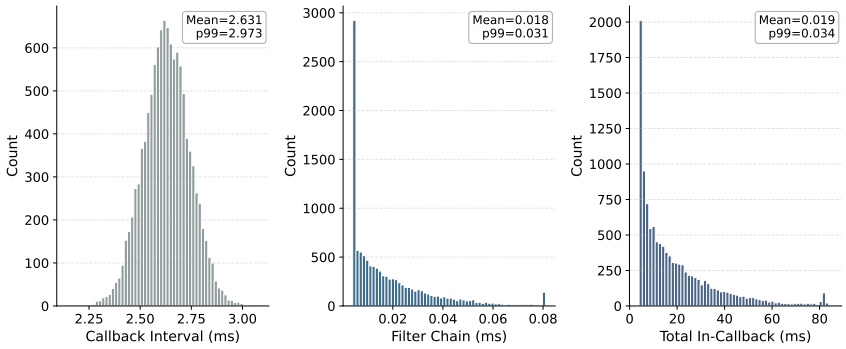

Figure 5.13: Latency distributions

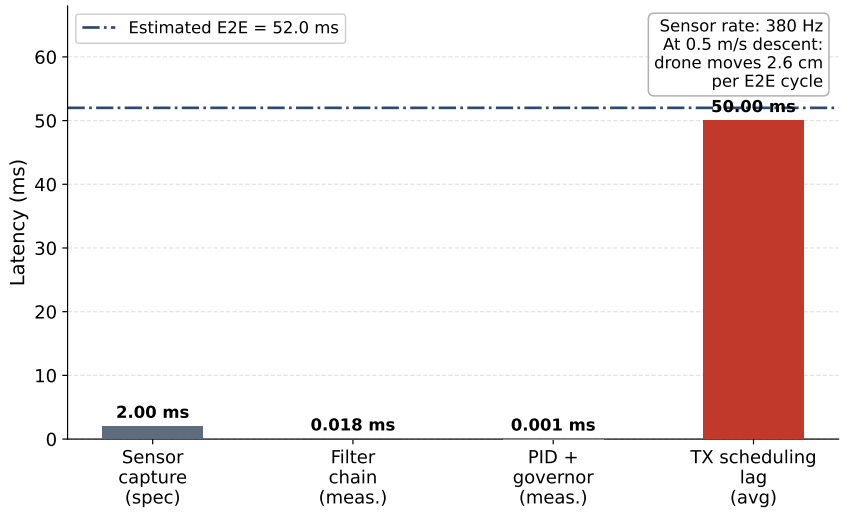

Figure 5.14: Per-stage latency breakdown

5.4.3 Discussion

5.4.3.1 Sensor update rate

The measured callback interval was 2.631 ms, corresponding to an effective sensor sampling rate of 380 Hz, while the manufacturer's nominal value is 400 Hz (2.5 ms). This 5 % difference is consistent with normal Linux kernel scheduling jitter on a non-real-time operating system and is not a limitation of the sensor itself. The highly concentrated distribution (standard deviation = 0.112 ms, as shown in the narrow histogram in Figure 5.13) confirms the highly stable callback time, with no noticeable frame drops observed during the 30-second test.

5.4.3.2 Algorithm computational cost

The complete in-callback computation—including five-sample median pre-filtering, chi-square outlier gating, one-dimensional Kalman filters (all applied to three axes), three-axis adaptive PID calculation, and descent rate regulator—takes an average of 0.019 ms (19 µs). The 99th percentile was 0.034 ms, confirming that even occasional cache misses or scheduling spikes do not significantly increase computation time.

This result is significant: the LattePanda companion computer consumes only 0.7 % of the available callback interval time in algorithm computation. The remaining 99.3 % of each 2.631 ms window can be used for other system tasks. Therefore, the platform performs far better than expected for this control workload, providing ample margin for future algorithm enhancements (such as IMU sensor fusion or extended state estimation) without concerns about timing violations.

5.4.3.3 TX scheduling lag

The primary latency component is the TX thread scheduling latency. A dedicated MAVLink TX thread runs at 10 Hz (100 ms period), distributing speed commands from shared atomic variables written by sensor callbacks. Therefore, the maximum possible latency between the algorithm output and its transmission is 100 ms. The expected value, evenly distributed over the 100 ms period, is 50 ms. This design was chosen to ensure the continuity of the MAVLink data stream during sensor data loss events and to meet the 115 200 baud serial bandwidth. A 50 Hz TX rate (∼20 ms maximum lag) is feasible if future deployments require lower command latency.

5.4.3.4 End-to-end pipeline latency

The estimated end-to-end latency from sensor frame acquisition to MAVLink command distribution is approximately 52 ms, including 2 ms of sensor acquisition latency (manufacturer specification), 0.019 ms of algorithm computation latency, and 50 ms of average TX thread scheduling latency. At a typical final approach descent rate of 0.5 m/s, this equates to a displacement of 2.6 cm between consecutive command cycles for the UAV. Thus, the 52 ms pipeline latency is well within the system's operational tolerance and will not constitute a performance bottleneck.

5.5 Hardware Integration Verification

Following the simulation validation, the full hardware stack was physically integrated and verified on the bench. The Sixdof Kipa optical sensor was connected to the LattePanda Mu companion computer via Gigabit Ethernet for data and USB for power, and the LattePanda was then connected to the Pixhawk 6C flight controller via USB. The complete MAVLink communication chain which started from the Kipa sensor through the LattePanda control application, via `mavlink-router`, at the end to the Pixhawk, was exercised with the real hardware components operating together.

As the final pre-flight verification milestone, the control application was commanded to execute the automated arming and OFFBOARD switching sequence described in Section 4.3. The LattePanda successfully established OFFBOARD mode on the Pixhawk 6C and issued the arm command, resulting in motor spin-up under algorithmic control. This demonstrates that the end-to-end command pipeline from optical position reading on the Kipa sensor, through the filtering and PID algorithm on the LattePanda, to motor actuation via the Pixhawk, is fully operational on the physical hardware.

However, the system has not yet proceeded to outdoor flight testing due to several outstanding mechanical and integration challenges. The LattePanda Mu Lite Carrier board and Kipa sensor are physically large relative to the drone frame, and their combined weight with the Pixhawk 6C and battery has not yet been confirmed within the drone's payload capacity. A custom 3D-printed mounting solution is required to attach the Kipa sensor, LattePanda, and Pixhawk securely to the airframe with appropriate weight distribution. Additionally, since the Kipa sensor must face downward toward the landing pad, physical standoffs or landing gear extensions are needed to prevent the sensor from contacting the ground during touchdown. These mechanical prerequisites, along with re-tuning of the PID gain parameters for the actual loaded drone dynamics, are identified as the immediate priorities for future work in Chapter 6.

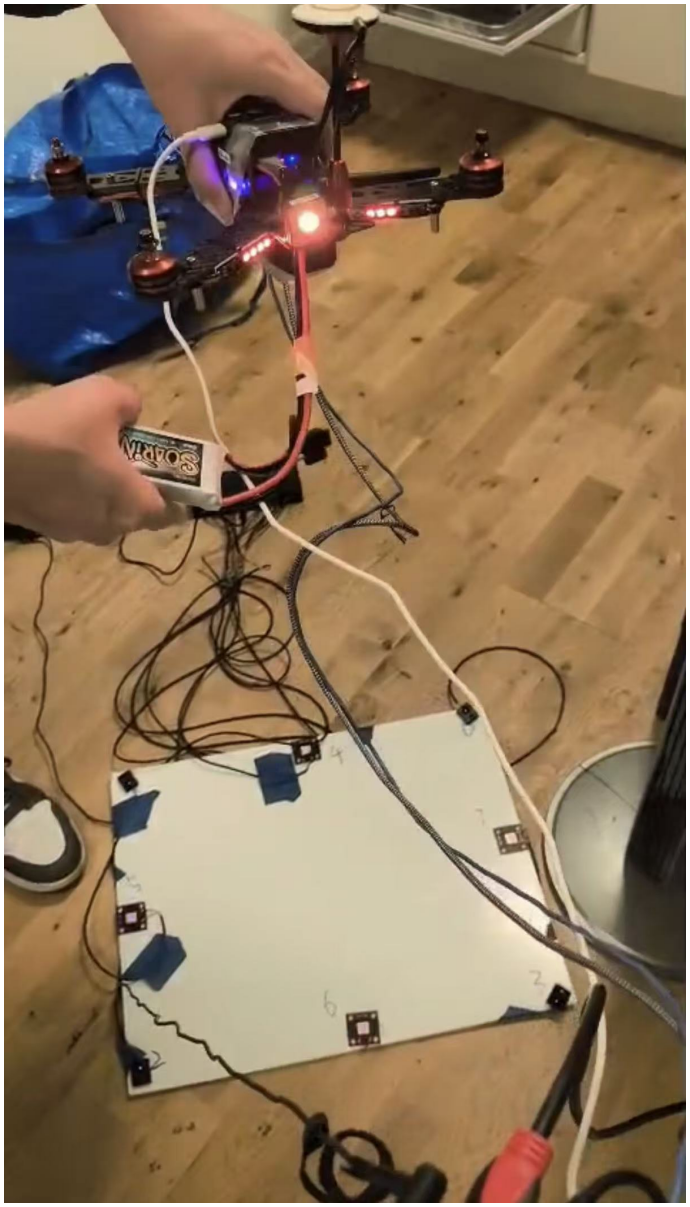

Figure 5.15: Bench integration test showing motor spin-up under control algorithm command

6
Conclusion

This project set out to design, implement, and evaluate a GPS-denied precision landing system for an autonomous UAV, motivated by the well-documented deficiencies of GNSS-based and passive vision-based landing in degraded environments. The "last meter problem" — the difficulty of achieving a safe, accurate touchdown on dynamic or GPS-denied platforms — was the central challenge addressed. An active infrared optical tracking system based on the Sixdof Space Kipa sensor was proposed and developed, with four primary objectives: end-to-end system functionality, a control algorithm robust to wind and adverse conditions, reliable safety behavior, and a pipeline latency below 75 ms.

6.1 Summary of Findings

6.1.1 Control Algorithm — Simulation Results

The control algorithm was evaluated in the ArduPilot SITL simulator across five experimental phases comprising 24 test configurations, with two important methodological corrections applied relative to prior simulation studies. The chi-squared outlier gate was updated with a consecutive-rejection counter to prevent permanent rejection of genuine drone displacement under sustained gusts. More significantly, the touchdown confirmation criterion was changed to evaluate altitude only, recording the drone's actual lateral position at ground contact rather than waiting for a favorable alignment moment. This produced larger but more honest error measurements that are representative of real deployment.

6.1.2 Baseline accuracy

Under ideal conditions (Phase 1), all nine runs achieved the *Excellent* classification with mean errors of 9.2–15.8 mm, more than 20 times below the 500 mm pass threshold. This confirms the mathematical correctness of the PID convergence and the discrete-time implementation.

6.1.3 Wind robustness

Phase 2 demonstrated that steady horizontal wind has minimal impact on landing precision. Mean error increased by only 2.6× across the full 0–15 m/s range, with all configurations classified as *Excellent* and landing times stable at approximately 35.6 s regardless of wind speed. The integral term effectively accumulated and

canceled the constant disturbance force, and the descent governor's cross-axis hold ensured the drone achieved lateral alignment before committing to final descent.

6.1.4 Turbulence robustness

Phase 3 identified turbulence as the dominant limiting factor. The system remained within the *Excellent* threshold at intensities 1 through 7, with mean errors rising from 49.9 mm to 223.9 mm. The failure boundary was clearly identified at intensity 9 ($\sigma \approx 9$ m/s), where the 100 ms command interval allows displacements approaching 0.9 m between successive Pixhawk updates, and simultaneous integral windup in the wrong direction prevents rapid recovery when the wind reverses.

6.1.5 Vertical wind behavior

Phase 4 varied the vertical elevation angle of the wind vector from $-45°$ (downdraft) to $+90°$ (pure updraft). All nine configurations achieved the *Excellent* classification. Brief `LOITER` events were observed when transient horizontal drift momentarily exceeded the `EXTREME` threshold, with the FSM correctly suspending descent and resuming automatically once the 40-sample RMS window recovered. The absence of `LOITER` in pure updraft reflects an inherent characteristic of the WeatherEstimator. It monitors horizontal XZ residuals, which vertical wind does not efficiently excite. The consistent accuracy across all elevation angles is also partly attributable to an ArduPilot SITL limitation which allows unlimited thrust to the drone. Physical flight testing is required to characterize the real thrust-limited behavior.

6.1.6 Combined worst-case performance

Phase 5 applied 15 m/s wind at DIR_Z = $+30°$ with turbulence intensity 5. All five runs confirmed touchdown, with mean error 181.4 ± 104.3 mm and four of five achieving *Excellent* classification. The high variance is intrinsic to the stochastic turbulence model rather than algorithmic instability, and the mean represents a $14.5\times$ increase over the no-wind baseline while remaining well within the pass threshold. Taken together, the simulation results confirm that the original control algorithm objective — landing within 0.5 m under wind up to 7 m/s without GPS — was exceeded. The algorithm tolerates steady wind well beyond 7 m/s and achieves sub-centimeter mean error under calm conditions. Turbulence intensity, rather than wind speed, is the primary constraint.

6.1.7 Kipa Sensor — Physical Range Characterization

The physical Kipa sensor was characterized across distances from 0.5 m to 12 m indoors and from 10 m to 23 m outdoors.
At 0.5 m and 1.0 m — the distances most critical to the touchdown decision — Y-axis depth errors were 2.7 cm and 2.8 cm. This consistent offset is attributable to a fixed discrepancy between the tape measure reference point and the sensor's internal depth reference rather than true inaccuracy; the net accuracy at these distances is sub-millimeter once the systematic offset is subtracted.

Indoor tracking was lost at approximately 13 m versus 23 m outdoors — a 43% range reduction attributed to IR multipath from room surfaces and near-IR background noise from household lighting. This indoor ceiling does not constrain the landing application since the final approach envelope lies well below this distance. Reliable outdoor tracking was confirmed at 10, 15, 20, and 23 m, covering the full descent profile supported by the beacon hardware.

6.1.8 Pipeline Latency Characterization

The sensor-to-command pipeline was characterized over 11,416 samples across a 30-second static test. The Kipa sensor delivered data at 380 Hz with a standard deviation of 0.112 ms, consistent with Linux scheduling jitter on a non-real-time OS. The complete in-callback computation — filtering, Kalman, PID, and descent governor — averaged 0.019 ms (19 μs), representing 0.7% utilization of the available callback window and demonstrating that the LattePanda provides substantial headroom for future algorithm additions.
The estimated end-to-end latency is 52 ms, dominated by the TX thread scheduling lag of 50 ms. This satisfies the 75 ms objective with margin. At a 0.5 m/s descent rate, the corresponding 2.6 cm displacement per command cycle is well within the system's operational tolerance.

6.1.9 System Integration

All hardware and software components were physically assembled and verified together on the bench. The automated arming and OFFBOARD switching sequence was successfully executed: the LattePanda established OFFBOARD mode on the Pixhawk 6C, issued the arm command, and achieved motor spin-up under algorithmic control. This confirms that the full command pipeline — from Kipa optical position reading through the filtering and PID computation, via MAVLink over `mavlink-router`, to PWM actuation by the Pixhawk — is fully operational on the physical hardware.
The system has not proceeded to outdoor flight testing during this project. Outstanding mechanical prerequisites include a custom 3D-printed mounting solution to attach the Kipa sensor, LattePanda Mu, and Pixhawk to the airframe with appropriate weight distribution, physical landing gear extensions to prevent the downward-facing sensor from contacting the ground during touchdown, and confirmation that the combined payload remains within the drone's thrust capacity. These items are the immediate prerequisites for transition to outdoor flight.

6.2 Limitations

6.2.1 Absence of flight testing

The most significant limitation is that the system has not been validated in actual flight. Simulation results provide strong evidence of algorithmic correctness and bench testing confirms hardware integration, but only a real closed-loop flight would

expose the complete system to genuine aerodynamic disturbances. The rotational dynamics of a real drone during descent subject the Kipa sensor to viewing angle changes that SITL does not replicate, and the actual payload capacity of the drone under the combined weight of all components remains unverified.

6.2.2 WeatherEstimator blind spot

The WeatherEstimator cannot detect correlated corruption of both its input sources. At turbulence intensity 9, the chi-squared gate and Kalman filter both tracked the same noisy signal, keeping their residual small and masking the true severity. This caused the PID to operate at full gain during conditions that warranted maximum gain reduction, contributing to the large XZ errors at touchdown. The WeatherEstimator also does not consider the Y axis residual which can make the system vulnerable to vertically changing wind.

6.2.3 Sensor range test methodology

The sensor range characterization used a discontinuous manual protocol that introduced angular misalignment errors between measurements. Y depth errors at mid-range (2-5 m) are therefore conservative estimates. Continuous tracking from altitude would produce smoother position estimates throughout the descent. Reducing the multipath effect in test is also important. However, in real flight test, the landing pad will be placed flat on the open space with no walls around it which means mutipath effect is not as notable as in the test.

6.3 Future Work

The immediate priority is completing the physical mounting solution: a 3D-printed bracket attaching the Kipa sensor (facing downward), LattePanda Mu, and Pixhawk to the airframe with balanced weight distribution, landing gear extensions providing adequate ground clearance for the sensor, and verification that total payload is within the drone's thrust budget. This is a prerequisite for all subsequent flight work.

The base PID gains were tuned in SITL on a simulated drone without the physical payload. Once the hardware is mounted, the drone's mass, moment of inertia, and hover throttle will change, requiring the gains to be re-characterized through the same empirical SITL-then-hardware process used in this project.

Following mechanical integration and gain re-tuning, a controlled outdoor flight test mirroring the SITL experiment design — fixed approach altitude, varied wind conditions, measured touchdown accuracy — would provide direct validation of the simulation predictions and close the loop between algorithmic and physical performance.

Adding the chi-squared gate's reset frequency as an additional severity indicator would address the correlated failure mode identified at turbulence intensity 9. If the gate resets more than a threshold number of times per second, the system would force

`EXTREME` classification and `LOITER` regardless of the positional residual, preventing full-gain operation during conditions the residual metric cannot detect.
Raising the MAVLink TX thread from 10 Hz to 50 Hz would reduce the average scheduling lag from 50 ms to 10 ms, lowering estimated end-to-end latency to approximately 12 ms and reducing the maximum command-interval displacement from 0.9 m to 0.18 m at $\sigma = 9$ m/s turbulence. This change requires verifying serial bandwidth and Pixhawk input buffer capacity at the higher rate.
A full 6DOF accuracy characterization at multiple outdoor distances — replacing the Y-only outdoor readings currently available — would complete the sensor performance baseline and quantify accuracy under direct sunlight, which operates near the 100,000 lux specification limit.
The Sixdof system's 6DOF tracking and 2.5 ms update interval make it architecturally suited for landing on a moving target such as a vehicle or ship deck. This use case would require the beacon map to be attached to the moving platform and the control algorithm extended to track the platform's velocity in addition to the drone's absolute position.

6.4 Closing Statement

This thesis has demonstrated that a GPS-denied precision UAV landing system based on active infrared optical tracking is technically feasible and performs to a high standard across a wide range of simulated environmental conditions. The four original objectives were met or exceeded: motor actuation through the full command pipeline was confirmed on the bench, the control algorithm surpassed the 0.5 m accuracy target with sub-centimeter mean error under calm conditions, the finite state machine and descent governor enforced safe behavior in every tested configuration without loss of control, and the 52 ms end-to-end pipeline latency satisfied the 75 ms requirement.
The Kipa sensor delivers sub-3 cmdepth accuracy and near-zero position noise in the critical final approach zone, and the LattePanda executes the complete filtering and control computation in under 20 µs per callback. The hardware stack has been verified to the pre-flight stage, with the remaining work being mechanical integration and outdoor flight validation rather than algorithmic or electronic development.
The most significant remaining step is field flight testing, which will close the loop between the simulation predictions and real aerodynamic behavior. The results presented here provide a well-characterized foundation for that next phase, and confirm that active infrared optical tracking is a compelling, all-weather alternative to GPS and passive visual landing systems for autonomous UAV precision landing applications.

Bibliography

[1] H. Shakhatreh *et al.*, "Unmanned aerial vehicles (UAVs): A survey on civil applications and key research challenges," *IEEE Access*, vol. 7, pp. 48 572–48 634, 2019.

[2] A. Gautam, P. B. Sujit, and S. Saripalli, "A survey of autonomous landing techniques for UAVs," in *Proc. Int. Conf. Unmanned Aircr. Syst. (ICUAS)*. IEEE, 2014, pp. 1210–1218.

[3] A. Gautam *et al.*, "Autonomous quadcopter landing on a moving target," *Sensors*, vol. 22, no. 3, 2022, art. no. 1116.

[4] L. Tavasci, F. Nex, and S. Gandolfi, "Reliability of real-time kinematic (RTK) positioning for low-cost drones' navigation across global navigation satellite system (GNSS) critical environments," *Sensors*, vol. 24, no. 18, 2024, art. no. 6096.

[5] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2011, pp. 3400–3407.

[6] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017.

[7] J. Krejsa and S. Vechet, "Infrared beacons based localization of mobile robot," *Elektronika ir Elektrotechnika*, vol. 117, no. 1, pp. 17–22, 2012.

[8] MAVLink. (2024) Protocol overview — MAVLink developer guide. Accessed: May 2026. [Online]. Available: https://mavlink.io/en/about/overview.html.

[9] A. Shah *et al.*, "Comparative analysis of median filter and its variants for removal of impulse noise from gray scale images," *J. King Saud Univ. — Comput. Inf. Sci.*, vol. 34, no. 3, pp. 505–519, 2022.

[10] K. Lee and E. N. Johnson, "Latency compensated visual-inertial odometry for agile autonomous flight," *Sensors*, vol. 20, no. 8, 2020, art. no. 2209.

[11] R. G. Brown and P. Y. C. Hwang, *Introduction to Random Signals and Applied Kalman Filtering with MATLAB Exercises*, 4th ed. John Wiley & Sons, 2012.

[12] Sixdof Space. (2026) Sixdof falcon development kit user guide, version 6. Accessed: May 2026. [Online]. Available: https://www.sixdofspace.com/downloads.

[13] ——. (2026) Sixdof system architecture. Accessed: May 2026. [Online]. Available: https://www.sixdofspace.com/_files/ugd/dcad42_7369f94d6d5b4d2e811fc6917adedd52.pdf.

[14] ——. (2026) Kipa sensor data sheet. Accessed: May 2026. [Online]. Available: https://www.sixdofspace.com/_files/ugd/dcad42_85e67a95c0ac4a7984361da4113817a0.pdf.

[15] ——. (2026) Beacon specifications data sheet. Accessed: May 2026. [Online]. Available: https://www.sixdofspace.com/_files/ugd/dcad42_96b29401a02240d8a4b652fbeb75598f.pdf.

[16] X. Ding *et al.*, "Experimental study of accuracy of high-rate GNSS in context of structural health monitoring," *Remote Sens.*, vol. 14, no. 19, 2022, art. no. 4989.

[17] LattePanda. (2026) LattePanda Mu — customizable micro x86 compute module. Accessed: May 2026. [Online]. Available: https://www.lattepanda.com/lattepanda-mu.

[18] ——. (2026) LattePanda Mu Lite Carrier documentation. Accessed: May 2026. [Online]. Available: https://docs.lattepanda.com/content/mu_edition/lite_carrier/.

[19] Holybro. (2026) Pixhawk 6c flight controller documentation. Accessed: May 2026. [Online]. Available: https://docs.holybro.com/autopilot/pixhawk-6c.

[20] ArduPilot. Mission planner simulation. Accessed: May 2026. [Online]. Available: https://ardupilot.org/planner/docs/mission-planner-simulation.html.

[21] QGroundControl. QGroundControl ground control station. Accessed: May 2026. [Online]. Available: https://qgroundcontrol.com.

[22] S. P. H. Driessen, N. H. J. Janssen, L. Wang, J. L. Palmer, and H. Nijmeijer, "Experimentally validated extended Kalman filter for UAV state estimation using low-cost sensors," *IFAC-PapersOnLine*, vol. 51, no. 15, pp. 43–48, 2018, 18th IFAC Symposium on System Identification SYSID 2018.

[23] K. Ogata, *Modern Control Engineering*, 5th ed. Prentice-Hall, 2010.

[24] S. Bouabdallah *et al.*, "PID vs LQ control techniques applied to an indoor micro quadrotor," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2004, pp. 2451–2456.

[25] A. Williams, *C++ Concurrency in Action*, 2nd ed. Manning Publications, 2019.

[26] PX4. (2024) Heartbeat/connection protocol — PX4 user guide. Accessed: May 2026. [Online]. Available: https://docs.px4.io/main/en/mavlink/protocols.

[27] K. J. Åström and B. Wittenmark, *Adaptive Control*, 2nd ed. Addison-Wesley, 1995.

[28] J. Springer, G. Þ. Guðmundsson, and M. Kyas, "A precision drone landing system using visual and IR fiducial markers and a multi-payload camera," 2024, arXiv:2403.03806. [Online]. Available: https://arxiv.org/abs/2403.03806.

DEPARTMENT OF MICROTECHNOLOGY AND NANOSCIENCE
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY