TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

Implementation of a UAV-Based Leader-Follower Convoy System Using Relative Localization

By:

Bikash Koirala (078/BAS/013)
Devis Maharjan (078/BAS/019)
Rikesh Poudel (078/BAS/035)
Riyaj Nepal (078/BAS/036)

A PROJECT REPORT SUBMITTED TO THE DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE BACHELOR'S DEGREE IN AEROSPACE ENGINEERING

DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING
LALITPUR, NEPAL

May, 2026

COPYRIGHT

TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

PULCHOWK CAMPUS

DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING

LETTER OF APPROVAL

The undersigned certify that they have read, and recommended to the Institute of Engineering for acceptance, a project report titled **"Implementation of a UAV-Based Leader-Follower Convoy System Using Relative Localization"** submitted by **Bikash Koirala, Devis Maharjan, Rikesh Poudel, and Riyaj Nepal** in partial fulfillment of the requirements for the Bachelor's Degree in Aerospace Engineering.

Asst. Prof. Biman Rimal
Supervisor
Dept. of Mechanical & Aerospace Engineering
IOE, Pulchowk Campus

Asst. Prof. Manoj Adhikari
Supervisor
Dept. of Mechanical & Aerospace Engineering
IOE, Pulchowk Campus

Asst. Prof. Bibek Dhungana
External Examiner
Institute of Engineering
Thapathali Campus

Asst. Prof. Dr. Sudip Bhattrai
Head of Department
Dept. of Mechanical & Aerospace Engineering
IOE, Pulchowk Campus

DATE OF APPROVAL: 2026-05-26

ABSTRACT

This project presents the design and validation of a leader-follower UAV convoy framework utilizing cloud-based synchronization for autonomous relative localization. The system architecture integrates Google Firebase for real-time data exchange, DroneKit for high-level mission guidance, and the ArduPilot flight stack for low-level stabilization. A distributed control strategy employing geodesic pursuit guidance and a hybrid hysteresis controller was implemented to manage formation flight at a 1 Hz update rate. System performance was validated through a Software-in-the-Loop (SITL) regime, synchronized to replicate the physical constraints of field trials, including a 1.15 m/s walking velocity and a 145 ms communication latency.

Experimental results demonstrate successful maintenance of a 4.5 m longitudinal offset and a 6.6 m vertical separation in the North-East-Down (NED) frame. The system achieved a mean Horizontal Tracking Error (HTE) of 0.167 m in simulation and 0.285 m in real-world deployment, with S-curve trajectory generation ensuring smooth motion and active stabilization. The analysis confirms that tracking deviations are primarily a deterministic function of network latency and quantization rather than algorithmic instability. This implementation establishes a validated platform for wide-area multi-agent operations and confirms the feasibility of cloud middleware as a reliable foundation for low-velocity autonomous convoy synchronization.

Keywords: Autonomous Flight, DroneKit, Geodesic Pursuit, Safety Shell, UAV Convoy

ACKNOWLEDGMENTS

This project is prepared in partial fulfillment of the requirements for the Bachelor's degree in Aerospace Engineering at the Institute of Engineering, Pulchowk Campus.

We express our sincere gratitude to our project supervisors, Asst. Prof. Biman Rimal and Asst. Prof. Manoj Adhikari, for their constant guidance and mentorship. Their technical expertise and constructive suggestions were invaluable to the successful completion of this work.

Our thanks go to Asst. Prof. Dr. Sudip Bhattrai, Head of the Department of Mechanical and Aerospace Engineering, and Asst. Prof. Kamal Darlami, Deputy Head, for their administrative support and encouragement. We further appreciate the Department for providing the academic environment necessary for this undertaking.

We extend our heartfelt thanks to Mr. Aniket Gupt for his exceptional technical guidance. We are also deeply grateful to Ameca Academy and Drone Sewa Nepal Pvt. Ltd. for their generous hardware support. Their contributions were vital in transitioning our theoretical designs into a functional system.

Finally, we thank our friends and colleagues for their direct and indirect support. This project has not only allowed us to implement the knowledge gained over the past four years but has also provided profound experience in professional collaboration and teamwork.

Authors:

Bikash Koirala (078/BAS/013)

Davis Maharjan (078/BAS/019)

Rikesh Poudel (078/BAS/035)

Riyaj Nepal (078/BAS/036)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS AND ABBREVIATIONS

API	Application Programming Interface
APF	Artificial Potential Field
BLDC	Brushless Direct Current
BS	Base Station
CAD	Computer-Aided Design
CAN	Controller Area Network
CoM	Center of Mass
CPU	Central Programming Unit
D1	Leader Drone
D2	Follower Drone
DCS	Distributed Control System
DOI	Digital Object Identifier
ECEF	Earth-Centered, Earth-Fixed
EKF	Extended Kalman Filter
EKF3	Extended Kalman Filter (Version 3)
ESC	Electronic Speed Controller
FMU	Flight Management Unit
GCS	Ground Control Station
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HDOP	Horizontal Dilution of Precision
HITL	Human-in-the-Loop
HO	Handover
HTE	Horizontal Tracking Error
IAT	Inter-Arrival Time
IMU	Inertial Measurement Unit

ISM	Industrial, Scientific, and Medical
JSON	JavaScript Object Notation
LiPo	Lithium Polymer
LOS	Line-of-Sight
LTE	Long-Term Evolution
MAVLink	Micro Air Vehicle Link
NED	North-East-Down
NLOS	Non-Line-of-Sight
NPR	Nepalese Rupee
OFDM	Orthogonal Frequency-Division Multiplexing
OODA	Observe-Orient-Decide-Act
PDR	Packet Delivery Ratio
PID	Proportional-Integral-Derivative
PWM	Pulse Width Modulation
QGC	QGroundControl
QoS	Quality of Service
RLF	Radio Link Failure
RMS	Root Mean Square
RMSE	Root Mean Square Error
RSRP	Reference Signal Received Power
RSRQ	Reference Signal Received Quality
RTDB	Realtime Database
RTL	Return-to-Launch
SAE	Society of Automotive Engineers
SBC	Single Board Computer
SDK	Software Development Kit
SITL	Software-in-the-Loop
SPOF	Single Point of Failure
SQL	Structured Query Language
TCP/IP	Transmission Control Protocol/Internet Protocol

UART	Universal Asynchronous Receiver-Transmitter
UAV	Unmanned Aerial Vehicle
UBEC	Universal Battery Eliminator Circuit
UDP	User Datagram Protocol
USB	Universal Serial Bus
V2V	Vehicle-to-Vehicle
VDOP	Vertical Dilution of Precision
WGS84	World Geodetic System 1984
WLAN	Wireless Local Area Network
XTE	Cross-Track Error

CHAPTER 1: INTRODUCTION

1.1 Background

The rapid advancement of Unmanned Aerial Vehicle (UAV) technology has facilitated a paradigm shift from isolated, single agent operations toward sophisticated multi agent coordination. Within this domain, leader-follower formation has emerged as a particularly effective architecture for enhancing operational capacity in complex missions such as aerial logistics, autonomous surveillance, and coordinated disaster response. Implementing a leader-follower architecture in UAV convoys provides a deterministic, computationally efficient solution for complex multi-agent coordination. By using relative localization rather than absolute global references for each agent, the system significantly reduces cumulative positioning errors and simplifies the control logic required for stable formation maintenance. This approach not only enhances mission reliability through hierarchical redundancy but also establishes a scalable framework that adapts to the dynamic constraints of real-world aerial logistics and autonomous surveillance missions. By implementing a "convoy" methodology, multiple UAS units function as a singular cohesive entity, which significantly reduces the cognitive burden on human operators while simultaneously increasing mission throughout and spatial coverage. This transition is not merely a matter of hardware scaling but involves the integration of complex control laws that ensure relative distance maintenance and aerodynamic safety. As these systems move from academic research into real world applications, the ability to maintain a stable formation during high speed transit has become a fundamental requirement for the next generation of autonomous aerial infrastructure.

Historically, the primary constraint governing these multi agent systems has been their reliance on short range Radio Frequency (RF) links for inter agent telemetry exchange. While traditional point to point protocols like 2.4 GHz or 915 MHz provide low-latency data transmission, they are inherently limited by physical distance and the requirement for

a clear Line of Sight (LOS).In geographically diverse or urban environments such as the mountainous terrain of Nepal or dense metropolitan centers signal occlusion and multipath interference frequently lead to catastrophic communication failure. These legacy systems are unable to support Non Line of Sight (NLOS) operations, effectively tethering the follower drone to a narrow radius around the leader. This "communication bottleneck" prevents the deployment of UAV convoys for long-range tasks like cross district medical delivery or extensive border patrolling, necessitating a move toward a more robust, range independent communication medium.

To address these systemic limitations, this project proposes a shift toward cellular connected telemetry and cloud-based synchronization. Using existing network infrastructure, the system decouples the physical distance between agents from their ability to exchange state vectors, enabling coordination over vast, geographically independent distances. The integration of the Google Firebase Realtime Database serves as an asynchronous jitter buffer, allowing the leader to publish its global coordinates calculated via the Haversine geodesic formula to a central cloud node that the follower can access in Realtime. This framework effectively transforms a local hardware to hardware problem into a scalable software to cloud architecture. By implementing a 1 Hz OODA (Observe, Orient, Decide, Act) loop supported by a 4 meter "Safety Shell" protocol, the system maintains formation integrity even amidst the stochastic latency inherent in cellular networks, providing a robust solution for beyond line-of-sight autonomous convoys.

1.2 Problem Statement

Autonomous leader-follower formation flight depends on two things working well together: accurate and timely knowledge of the leader's position, and a control strategy that can translate that knowledge into safe and stable follower behavior. Achieving both simultaneously over a network-based communication channel is a non-trivial challenge. When inter-agent state exchange is routed through a shared Wi-Fi network and a cloud database, communication introduces stochastic and non-deterministic latency driven by packet contention, variable database write times, and network congestion. If not explicitly

managed, these delays can cause the follower to act on outdated leader state data, threatening both formation stability and collision avoidance. At the same time, operating the follower at a low positional update frequency, which is a practical necessity given the nature of cloud synchronization, creates a gap between the discrete guidance commands arriving from the cloud and the continuous physical dynamics of the vehicle. Bridging that gap without producing jerky or oscillatory motion requires careful trajectory generation. Beyond communication, accurate relative localization over GPS coordinates presents its own challenge. Simple Euclidean subtraction of latitude and longitude values introduces geometric errors that grow with separation distance, making it unsuitable for precise formation control. A geodesically accurate distance computation is necessary to ensure that the guidance law operates on a true representation of the spatial relationship between agents. There is therefore a clear need for a convoy system that accurately localizes the follower relative to the leader using geodesic methods, tolerates network-induced delays through robust control design, enforces hard geometric safety boundaries, and produces smooth autonomous follower behavior from low-frequency cloud updates. This project directly addresses that need through the design, implementation, and real-world validation of such a system.

1.3 Objectives

1.3.1 Main Objective

To implement and validate a leader-follower UAV convoy system that uses Wi-Fi connectivity and Google Firebase to achieve autonomous relative localization and formation flight.

1.3.2 Specific Objectives

i) To implement a geodesic based pursuit logic for accurate relative localization between the leader and follower UAVs

ii) To establish a cloud mediated telemetry link using the Google Firebase Realtime

Database for real time state exchange over Wi-Fi monitored through a VNC remote display on a ground station laptop.

iii) To validate the system through distributed SITL simulations and real-world flight trials in which the follower drone physically tracks the manually carried leader drone.

1.4 Scope of the Project

The scope of this project covers the implementation of a distributed control architecture where high-level mission intelligence is separated from low-level flight stabilization. The work focuses on developing Python-based guidance scripts, integrating cloud middleware for telemetry exchange, and verifying system logic within a physics-based simulation environment. The framework is evaluated through Software-in-the-Loop (SITL) simulations running on a distributed two-node Ubuntu network, and extends to real-world hardware testing. This involves the integration of the Pixhawk flight controller and Raspberry Pi, connected via serial communication using the MAVLink protocol, to acquire and transmit live GPS positional data from the leader drone to Google Firebase over Wi-Fi. Ground station monitoring is handled through a VNC remote display on a laptop. The follower drone is then validated in real-world flight trials where it physically tracks the manually carried leader drone, confirming the end-to-end behavior of the convoy system under actual operating conditions.

1.5 Limitations

Despite the robust design, certain limitations apply to this project:

i) The current system architecture is optimized for a leader follower pair and does not yet support large scale decentralized swarm expansion.

ii) Collision avoidance is restricted to inter agent safety; secondary obstacle avoidance, including buildings, power lines, or terrain, is not implemented in the current scope.

iii) The system reliability is strictly dependent on the availability and signal strength of existing cellular network(Wifi or 4G LTE) infrastructure.

iv) The coordination logic is fundamentally dependent on Global Positioning System (GPS) availability; the framework does not support autonomous operation in GPS denied environments or areas with significant GNSS signal degradation.

v) The leader drone does not operate under its own propulsion during real-world testing; it is carried manually, meaning the system has not been validated under conditions where both agents are simultaneously airborne.

1.6 System Requirements (For Development/Implementation)

This section identifies the fundamental hardware and software components required to develop and implement the leader-follower convoy system.

1.6.1 Hardware Requirements

The hardware architecture is based on a modular "Companion Computer" design, which separates real time stabilization from high level mission intelligence:

I. **Flight Control System:** At the core of the avionics stack is the Pixhawk 4, a 32 bit advanced autopilot running ArduCopter 4.x firmware. It is responsible for low-level motor control, reading inertial sensors at 400Hz, and performing essential state estimation. To achieve the global position accuracy required for formation algorithms, it is integrated with a u blox Neo M8N GPS module. The Pixhawk provides the necessary stability and safety protocols, such as S curve trajectory generation, which ensures smooth transitions between the discrete 1 Hz updates received from the cloud.

II. **Companion Computer:** A microprocessor based unit integrated into the drone frame to run high level guidance scripts, perform geodesic calculations, and manage

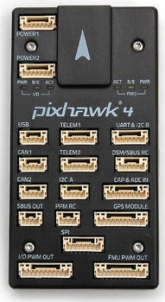


Figure 1.1: Pixhawk 4 Flight Controller used for Low-Level Flight Control

network connectivity.To enable autonomous decision making and range independent coordination, a Raspberry Pi 4 is integrated into each drone's frame as a dedicated companion computer. Acting as the system's high level intelligence, it executes Python based guidance scripts through the DroneKit library, manages the Wifi connection for cloud synchronization, and performs complex geodesic calculations such as the Haversine formula to ensure precise relative localization between the leader and follower. The unit interfaces with the Pixhawk 4 flight controller via a UART serial connection on the TELEM1 port, utilizing the MAVLink protocol to maintain high throughput data exchange between the autonomous mission logic and the underlying flight firmware.

III. **Wifi Communication Module:** A cellular modem interface allowing the companion computer to access the internet for cloud-based telemetry exchange.Unlike traditional drones limited by short-range radio frequency links, this system utilizes Wifi (scalable to 4G LTE USB Modem). This allows the companion computer to access the internet and treat the cellular network as a standard Ethernet

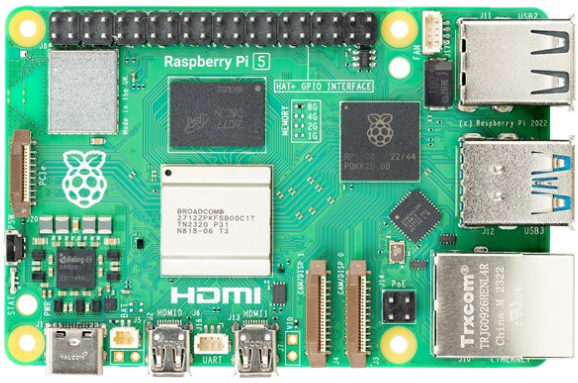


Figure 1.2: Raspberry Pi 5 used as the Onboard Companion Computer

interface. This enables the drones to exchange state vectors through the Google Firebase Realtime Database, facilitating stable formation flight without requiring a direct line of sight telemetry link.

IV. **Propulsion System:** The drone utilizes a generic Quadcopter X-configuration frame powered by four high performance Brushless DC motors paired with high surface area propellers to ensure aerodynamic efficiency and minimize vibrations for improved sensor accuracy. These motors are managed by Electronic Speed Controllers that regulate speeds via PWM signals from the Pixhawk flight controller.This setup ensures precisely synchronized thrust delivery, which is critical for maintaining the 10 meter vertical offset between vehicles, thereby preventing instability caused by the leader drone's downwash turbulence during convoy operations.Comprises Brushless DC motors and high surface area propellers selected to provide stable thrust and aerodynamic efficiency.

V. **Electronic Speed Controllers (ESCs):** These units regulate motor speeds based on

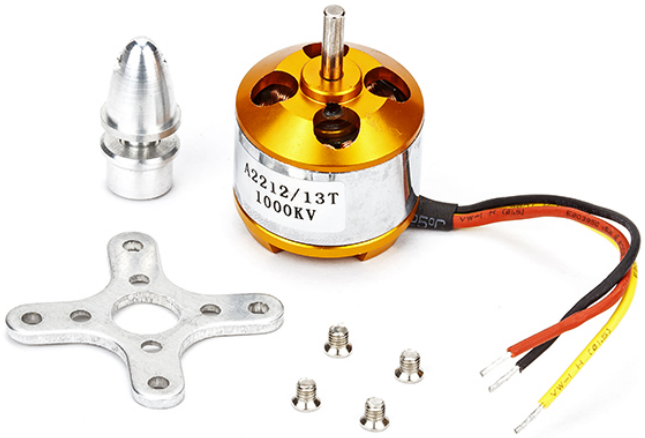


Figure 1.3: A2212 1000KV BLDC Motor utilized for propulsion.

PWM signals from the Pixhawk. They are calibrated to ensure synchronized thrust delivery, which is critical for maintaining the 10 meter vertical offset required to avoid leader downwash turbulence.

Figure 1.4: Electronic Speed Controller (ESC)

VI. **Power Distribution:** A Lithium Polymer (LiPo) battery source with isolated power rails to ensure regulated voltage for the avionics and high-current supply for the motors.

Figure 1.5: LiPo Battery

1.6.2 Software Requirements

The software environment is built on open source frameworks to ensure interoperability:

I. **Operating System:** A Linux-based environment for the development machines and companion computers.It provides a robust, open-source development environment necessary for running high-level robotic middleware. Its native support for Python environments and network-side processing makes it the ideal choice for both the development workstation and the onboard companion computer.It handles the critical flight tasks and interprets high-level commands into precise motor outputs.

II. **Flight Firmware:** ArduPilot runs on the flight controller to manage core navigation and stabilization tasks, providing the high level EKF3 state estimation and low level motor control required for precise trajectory execution.

III. **Simulation Stack:** The Gazebo physics engine is combined with ArduPilot SITL to provide a high-fidelity virtual environment for functional validation, allowing the

testing of convoy logic under simulated network latency.

IV. **Programming Framework:** Python 3 is employed as the primary programming language, utilizing the DroneKit library to implement high-level autonomous mission control, including the geodesic pursuit guidance and the 1 Hz control loop.

V. **Cloud Infrastructure:** Google Firebase Realtime Database is integrated as the cloud middleware to facilitate asynchronous telemetry synchronization between the leader and follower UAVs over Wifi (or 4G LTE cellular networks).

VI. **Communication Protocol:**MAVLink is utilized as the lightweight, bidirectional messaging protocol for seamless serial data exchange between the flight controller's hardware and the onboard companion computer.

CHAPTER 2: LITERATURE REVIEW

Autonomous convoying, also referred to as autonomous platooning, involves a group of vehicles traveling in a coordinated formation to optimize logistics, surveillance, and defense operations [1]. As research shifts from ground based systems [2] to the aerial domain, the technical focus has pivoted toward managing high dynamic three dimensional coordination, long range communication, and aerodynamic wake interference. This review synthesizes current research across these critical pillars.

2.1 Theoretical Foundations of Autonomous Convoying

The primary challenge in aerial coordination is the maintenance of a stable geometric relationship between agents. Ali et al. define the leader-follower rigid topology as a system where a follower's control input is derived entirely from the leader's state [3]. This unidirectional information flow, as illustrated in our methodology (see Figure 4.3), allows for scalable coordination but requires robust hybrid controllers to manage transition states [3]. A critical aerodynamic constraint in rotorcraft formation is downwash interference. Shim et al. demonstrate that as a leader UAV generates lift, it creates a high-velocity downward air column; if a follower enters this wake, it suffers significant thrust loss [4]. McKiernan and Glauser suggest that maintaining a strict vertical offset is the most effective mitigation strategy [5]. Our implementation of a fixed 10-meter vertical bias, detailed in the Inner-Loop Control Architecture (Figure 4.6), directly addresses these empirical findings to ensure aerodynamic stability.

2.2 Relative Localization Paradigms

A cornerstone of stable formation flight is the ability of a follower agent to determine its state vector relative to the leader. Kilic and Yang categorize localization into two primary paradigms: absolute-referenced and relative-vectoring systems [6]. While vision-based

systems using Aruco markers or optical flow provide high precision at close ranges, they are susceptible to environmental occlusion and lighting variability [6].

In contrast, GNSS-based relative localization where the follower calculates a relative offset based on synchronized GPS coordinates is better suited for long-range logistics and "convoy" missions in open environments [7]. Our research utilizes this GNSS-vectoring approach, leveraging the high-altitude visibility of the Pulchowk football ground to maintain a stable horizontal tracking envelope without the need for line-of-sight visual sensors.

2.3 Communication Protocols and Telemetry Standards

The reliability of a leader-follower convoy depends on the efficient marshaling of telemetry data between the onboard computer (Raspberry Pi) and the flight controller (Pixhawk). Meier et al. introduced the MAVLink (Micro Air Vehicle Link) protocol as a lightweight, LGPL-licensed message marshaling library designed for high-vibration, low-bandwidth environments [8].

Unlike heavier middleware such as the Robot Operating System (ROS), MAVLink uses a "point-to-point" and "publish-subscribe" architecture that minimizes the computational footprint on the agent [8]. This efficiency is critical for our implementation, as it allows the Raspberry Pi to simultaneously manage the Firebase cloud-sync and the real-time conversion of geodesic coordinates into MAVLink `SET_POSITION_TARGET_LOCAL_NED` commands.

2.4 Cloud Middleware and Asynchronous Synchronization

Managing the data exchange between independent drones across a cellular network requires an efficient middleware layer. Patnaik identifies Google Firebase as a superior choice for real time UAV applications due to its WebSocket-based persistent connections, which eliminate the high overhead of standard HTTP polling [9]. Maranco et al. further

argue that cloud hosted Realtime Databases (RTDB) act as an asynchronous "jitter buffer," decoupling the transmission frequency of the leader from the retrieval frequency of the follower [10]. The hierarchical JSON structure, as depicted in Figure 4.9, allows for flexible state-vector updates without the rigid constraints of traditional SQL databases [11].

2.5 Network Reliability and the Age of Information (AoI)

Operating a convoy over a cellular (LTE) cloud link introduces stochastic delays known as network jitter. Hespanha et al. describe the "Networked Control System" (NCS) as a framework where control loops are closed over a communication channel, making stability dependent on the Round-Trip Time (RTT) [12].

A critical metric in such systems is the *Age of Information* (AoI), which quantifies the time elapsed since the leader's most recent state update was generated [13]. In leader-follower systems, high AoI can lead to "string instability," where tracking errors amplify down the chain of followers. Our study specifically analyzes the 145 ms cloud-synchronization RTT to justify the use of safety hysteresis and S-curve smoothing as a means to mitigate the impact of stale information on formation stability.

2.6 Mathematical Models for Geodesic Navigation

Precision in long-range convoying requires a transition from planar Euclidean geometry to terrestrial manifolds. Sinnott argues that the Haversine formula is the "virtuous" method for distance calculation because it remains numerically stable even at the very small angular separations typical of UAV convoys [14]. Jia further explains that geodesics represent the energy optimal "Great Circle" paths across the Earth's oblate spheroid surface [15].

By combining these geodesic distance calculations with Line-of-Sight (LOS) Pure Pursuit guidance as visualized in Figure 4.4 and Figure 3.4 the system becomes resilient to magnetometer drift and directional sensor noise prevalent in urban "Urban Canyons" [16].

2.7 Trajectory Smoothing and Control Stability

A significant hurdle in discrete control (1 Hz updates) is the potential for jerky, "stop and go" motion. Biagiotti and Melchiorri explain that jerk limited trajectory planning, which constrains the third derivative of position, is essential for smooth kinematic transitions [17]. The ArduPilot S-Curve generator, shown in Figure 4.11, utilizes these principles to convert discrete waypoints into continuous acceleration profiles [18]. This ensures that mechanical stress is minimized while the follower glides through the 1 second interval between cloud updates.

2.8 Simulation and Validation Frameworks

The high cost and risk of multi-UAV field testing necessitate rigorous simulation validation. Koenig and Howard established Gazebo as the gold standard for simulating rigid-body dynamics and stochastic sensor noise [19]. ArduPilot Software in the Loop (SITL) allows for the validation of the exact same firmware that will be deployed on the physical Pixhawk hardware [20]. To avoid simulation desynchronization caused by CPU bottlenecks, Fabra et al. recommend a distributed architecture where each UAV agent resides on a dedicated computer [21]. Our use of a distributed network topology, illustrated in Figure 4.13, ensures that the simulated LTE performance and physics timing remain realistic and validated.

CHAPTER 3: THEORETICAL BACKGROUND

This chapter establishes the mathematical and physical principles required to implement a robust leader-follower UAV convoy. It covers the foundations of geodesy, 3D kinematics, control theory, and the stochastic modeling of cellular-based communication networks. The coordinate frame hierarchy is illustrated in Figure 3.3.

3.1 Quadcopter Dynamics

The dynamics of a quadcopter represent a complex multi-variable system involving aerodynamic forces and rigid-body mechanics. To model the system effectively for a leader-follower convoy, the UAV is treated as a six-degree-of-freedom (6-DOF) rigid body. To simplify the mathematical formulation, several assumptions are maintained: the motor resistance is negligible, the airframe is symmetric and rigid, blade flapping is ignored, environmental wind velocities are negligible, and the center of gravity (CoG) is coincident with the origin of the body-fixed frame [22].

The state is observed through two primary frames: the Inertial Frame (NED: North-East-Down), which is stationary relative to the ground, and the Body-Fixed Frame, which is attached to the UAV's CoG and moves with the vehicle.

Figure 3.1: Isometric View of Quadcopter

3.1.1 Rotor Dynamics

Each rotor generates a thrust force F_i and a reactive torque τ_i proportional to the square of the angular velocity ω_i. This relationship is simplified as [23]:

$$F_i \approx k\omega_i^2, \quad \tau_i \approx b\omega_i^2 \tag{3.1}$$

where k is the thrust factor and b is the drag constant determined by blade geometry and air density. The total thrust T_B in the body frame and the resulting moments for roll (τ_ϕ), pitch (τ_θ), and yaw (τ_ψ) are calculated based on the arm length L and the differential thrust between motor pairs.

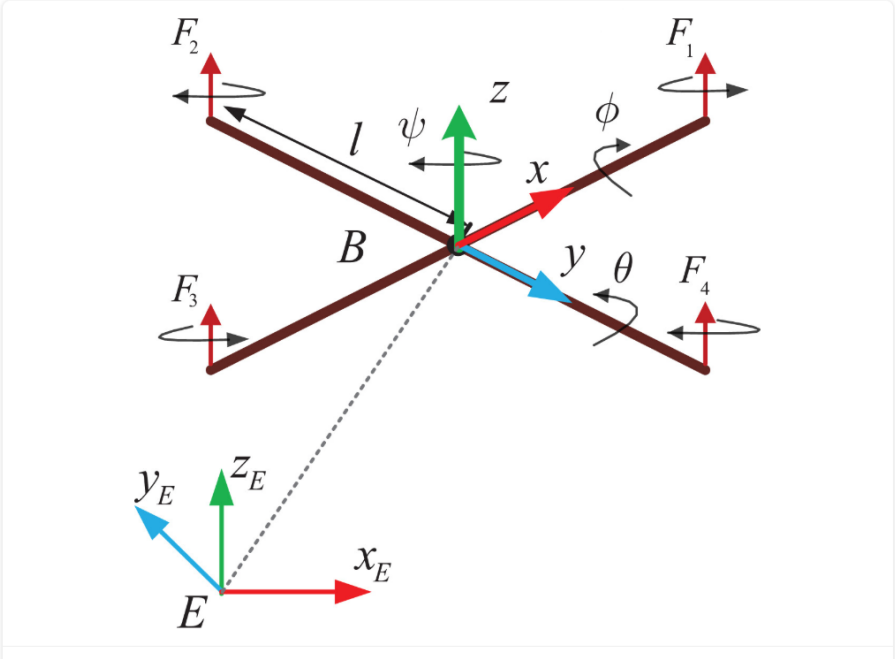


Figure 3.2: Rotor Dynamics.

3.1.2 Kinematics and Coordinate Transformation

To translate high-level guidance commands from the inertial frame to the vehicle's flight computer, a rotation matrix is required. The transformation from the body-fixed frame to the inertial frame is achieved using XYZ Euler angles: roll (ϕ), pitch (θ), and yaw (ψ). The resulting rotation matrix R_I is defined as [24]:

$$R_I = \begin{bmatrix} C_\psi C_\theta & C_\psi S_\theta S_\phi - S_\psi C_\phi & C_\psi S_\theta C_\phi + S_\psi S_\phi \\ S_\psi C_\theta & S_\psi S_\theta S_\phi + C_\psi C_\phi & S_\psi S_\theta C_\phi - C_\psi S_\phi \\ -S_\theta & C_\theta S_\phi & C_\theta C_\phi \end{bmatrix} \tag{3.2}$$

where C and S represent cos and sin, respectively.

3.1.3 Equations of Motion

Following the Newton-Euler formulation, the translational and rotational dynamics of the quadcopter are defined by integrating Newton's second law with Euler's equations for rigid bodies [23]. The translational motion in the inertial frame is governed by:

$$m\ddot{x} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + F_D + R_I T_B \tag{3.3}$$

where m is the mass, g is gravity, and F_D represents the aerodynamic drag forces. The rotational acceleration is derived from Euler's rotational equation:

$$I\dot{\omega} + \omega \times (I\omega) = \tau \tag{3.4}$$

Assuming a symmetrical diagonal inertia matrix $I = \text{diag}(I_{xx}, I_{yy}, I_{zz})$, this allows the system to calculate the necessary angular velocities required to maintain stability during pursuit maneuvers.

3.1.4 Aerodynamics of Translational Flight and Thrust Vectoring

For a multicopter to transition from a stationary hover to steady-state translational flight, it must reorient its total thrust vector to overcome aerodynamic resistance. Unlike fixed-wing aircraft that utilize control surfaces for lift and propulsion separation, a quadcopter utilizes thrust vectoring by tilting the entire airframe.

3.1.4.1 Force Equilibrium in Forward Flight

In a steady-state forward flight at a constant velocity (v), the vehicle must maintain an equilibrium between four primary forces: Thrust (T), Weight (W), Drag (D), and Lift (L). For small-scale UAVs at low velocities, the lift generated by the frame is often negligible,

and the equilibrium is defined by the following equations in the longitudinal plane:

$$\sum F_z = T\cos\theta - W = 0 \implies T\cos\theta = mg \tag{3.5}$$

$$\sum F_x = T\sin\theta - D = 0 \implies T\sin\theta = \frac{1}{2}\rho v^2 C_D A \tag{3.6}$$

where θ represents the pitch angle, m is the mass of the vehicle, ρ is the air density, C_D is the parasitic drag coefficient, and A is the frontal projected area of the S500 frame.

3.1.4.2 The Pitch-Velocity Relationship

By dividing the horizontal equilibrium by the vertical equilibrium, the fundamental relationship between the required pitch angle and the translational velocity is derived:

$$\tan\theta = \frac{D}{W} = \frac{\rho v^2 C_D A}{2mg} \tag{3.7}$$

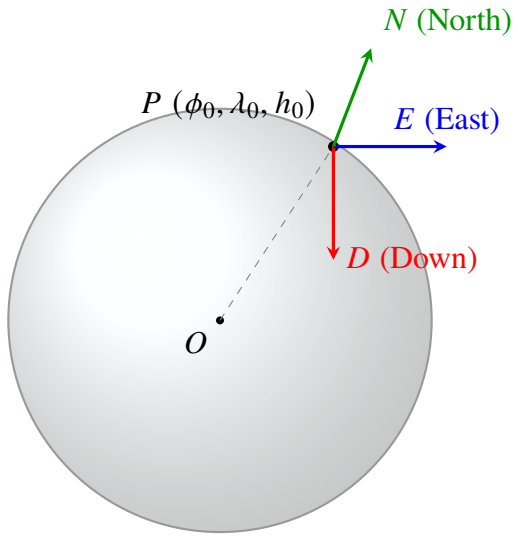
This equation implies that for a specific airframe and mass, the pitch angle θ is a deterministic function of the velocity v. In this study, the observed pitch of approximately 2.8° at a velocity of 1.15 m/s serves as a physical validation that the autopilot's control loop is correctly balancing the parasitic drag of the S500 frame against its weight to maintain convoy synchronization [25].

3.1.4.3 Small Angle Approximation in Stabilization

Given that the mission velocities for the ground-based convoy trials are relatively low ($v < 2$ m/s), the required tilt angles remain within the small-angle approximation regime ($\sin\theta \approx \theta$ in radians). This allows the linearized PID controllers within the ArduPilot flight stack to maintain high stabilization authority while simultaneously providing the necessary horizontal thrust component for pursuit [26].

3.2 Coordinate Systems and Geodesy

Navigation for autonomous aerial vehicles requires precise transformations between global planetary models and local maneuverable frames [27]. Figure 3.3 provides a visual summary of the two reference frames discussed below and their relationship.



WGS84 Ellipsoid with local NED tangent frame at P

Figure 3.3: Relationship between the global WGS84 geodetic frame and the local North-East-Down (NED) tangent plane at a home point P.

3.2.1 World Geodetic System (WGS84)

The Global Positioning System (GPS) utilizes the WGS84 ellipsoidal model as its reference surface. A position is defined by geodetic coordinates: Latitude (ϕ), Longitude (λ), and Altitude (h), where ϕ is the angle north or south of the equator, λ is the angle east or west of the prime meridian, and h is the height above the reference ellipsoid. While accurate for global tracking, these coordinates are non-linear; the physical distance represented by one degree of longitude varies as a function of the cosine of the latitude [15].

3.2.2 Local Tangent Plane: NED Frame

For high-frequency inter-agent coordination, geodetic coordinates are transformed into a local North-East-Down (NED) Cartesian frame. This frame assumes a flat-earth tangent at a specific home point (ϕ_0, λ_0, h_0), as shown in Figure 3.3. The NED frame is the standard for localized UAV control as it simplifies the calculation of velocity vectors and distance offsets.

For short-range maneuvers where the Earth's curvature is negligible, the geodetic-to-NED conversion can be linearized as follows:

$$\Delta N = R \cdot (\phi - \phi_0) \tag{3.8}$$

$$\Delta E = R \cdot (\lambda - \lambda_0) \cdot \cos(\phi_0) \tag{3.9}$$

where ΔN is the northward displacement in meters, ΔE is the eastward displacement in meters, R is the Earth's mean radius ($R \approx 6{,}371{,}000$ m), ϕ and λ are the current latitude and longitude in radians, and ϕ_0 and λ_0 are the latitude and longitude of the NED origin (home point) in radians.

This linearized approximation is valid for relative displacements on the order of a few kilometers, well within the operational envelope of a close-formation UAV convoy. For the inter-agent separations encountered in this project (typically < 50 m), the approximation error is negligible compared to GPS measurement uncertainty (≈ 2.5 m CEP).

The complete linearized geodetic-to-NED transformation, including the Down axis, can be expressed in compact vector–matrix form as:

$$\begin{bmatrix} \Delta N \\ \Delta E \\ \Delta D \end{bmatrix} = \begin{bmatrix} R & 0 & 0 \\ 0 & R\cos\phi_0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} \phi - \phi_0 \\ \lambda - \lambda_0 \\ h - h_0 \end{bmatrix} \tag{3.10}$$

where $\Delta D = -(h - h_0)$ is the downward displacement (positive toward the Earth's center), h is the current altitude above the WGS84 ellipsoid, and h_0 is the altitude of the NED origin. The negative sign convention on the Down axis follows the aerospace standard: positive D points toward the Earth, so an increase in geometric altitude corresponds to a decrease in D. This compact linearized transformation is the standard formulation presented by Stevens et al. [22] and forms the basis for all local-frame velocity and distance computations used by the follower's guidance logic.

3.3 Geospatial Kinematics and Distance Estimation

The operational stability of a leader-follower UAV convoy is fundamentally anchored in the precision of the relative distance vector, $\mathbf{r}_{rel}$, maintained between the agents. In an autonomous formation, the follower does not operate in a vacuum; its primary objective is to regulate its state based on its perceived position relative to the leader. Consequently, any error in estimating this vector directly translates into tracking instability or, in the worst-case scenario, a failure of the safety shell logic.

3.3.1 Defining the Relative State Vector

At any discrete time step t, the relative distance vector $\mathbf{r}_{rel}$ serves as the primary feedback signal for the guidance controller. It is formally defined as the difference between the leader's global state $\mathbf{X}_L$ and the follower's global state $\mathbf{X}_F$:

$$\mathbf{r}_{rel}(t) = \mathbf{X}_L(t) - \mathbf{X}_F(t) \tag{3.11}$$

In practice, this vector is rarely a static value. It is a dynamic estimation derived from the

fusion of high-frequency inertial measurements and lower-frequency global positioning data. The accuracy of this estimation is vital, as even sub-meter discrepancies can cause the controller to perceive a non-existent threat, leading to oscillatory braking maneuvers or mechanical stress on the actuators.

3.3.2 Geospatial Constraints and Frame Transformation

While simple Euclidean subtraction is sufficient for small-scale robotics, aerial convoys operating over global coordinates must account for the Earth's geometry. Longitude and latitude represent a non-linear coordinate system where the physical distance of one degree varies based on the current latitude.

To resolve this, the system must perform a two-step kinematic transformation. First, the geodetic coordinates are projected onto a local North-East-Down (NED) Cartesian frame. This projection creates a local tangent plane where standard geometric laws apply, allowing for high-frequency control updates that are computationally efficient.

3.3.3 Numerical Stability in Distance Estimation

To maintain the horizontal component of the formation, the Haversine formula is employed. Unlike the simpler Spherical Law of Cosines, the Haversine method is specifically chosen for its numerical stability. When agents are in close proximity—typical of a 5m to 10m convoy—the small angular separations can lead to catastrophic cancellation errors in standard trigonometric functions.

The horizontal distance d_{2D} is calculated to account for the Earth's curvature, ensuring that the guidance logic remains consistent regardless of the global location:

$$a = \sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_L)\cos(\phi_F)\sin^2\left(\frac{\Delta\lambda}{2}\right) \tag{3.12}$$

$$d_{2D} = 2R \cdot \text{atan2}\left(\sqrt{a}, \sqrt{1-a}\right) \tag{3.13}$$

By establishing this rigorous geospatial framework, the follower can derive a stable tracking error signal. This allows the low-level PID loops to respond to real-world motion rather than sensor noise or geometric approximations, ensuring the formation remains "string stable" even during high-velocity maneuvers.

3.3.4 The Haversine Formula

To calculate the 2D "great-circle" distance (d_{2D}) between two GPS points, the Haversine formula is employed to account for the Earth's curvature. This method is numerically stable even at the small angular separations typical of UAV convoys [28]. The distance is calculated as:

$$a = \sin^2\left(\frac{\phi_L - \phi_F}{2}\right) + \cos(\phi_L)\ \cos(\phi_F)\ \sin^2\left(\frac{\lambda_L - \lambda_F}{2}\right) \tag{3.14}$$

$$d_{2D} = 2R \cdot \text{atan2}\left(\sqrt{a},\ \sqrt{1-a}\right) \tag{3.15}$$

where ϕ_L and ϕ_F are the latitudes of the Leader and Follower in radians, λ_L and λ_F are the longitudes of the Leader and Follower in radians, R is the Earth's mean radius ($R \approx 6{,}371{,}000$ m), a is the square of the half-chord length between the two points, and d_{2D} is the resulting great-circle distance in meters [28].

3.3.5 Numerical Stability of the Haversine Formula

The selection of the Haversine formula over the more conventional Spherical Law of Cosines is not arbitrary but is motivated by a critical *numerical stability* requirement. The Law of Cosines computes the great-circle distance as:

$$d = R \cdot \arccos\big(\sin\phi_L \, \sin\phi_F + \cos\phi_L \, \cos\phi_F \, \cos(\Delta\lambda)\big) \tag{3.16}$$

For the small angular separations typical of a UAV convoy ($\Delta\phi, \Delta\lambda \ll 1$), the argument of the arccos function approaches 1.0. In IEEE 754 double-precision floating-point arithmetic, this leads to *catastrophic cancellation*: the subtraction $(1 - \cos\theta)$ for small θ annihilates significant digits, producing an output dominated by rounding error [29].

The Haversine reformulation avoids this pitfall entirely. By rewriting the distance computation in terms of $\sin^2(\theta/2)$ — the "half-versed-sine" — the critical subtraction is replaced by a squaring operation, which preserves numerical precision for arbitrarily small angles. Specifically, the identity:

$$\mathrm{hav}(\theta) = \sin^2\left(\frac{\theta}{2}\right) = \frac{1 - \cos\theta}{2} \tag{3.17}$$

maps the problematic range $\cos\theta \approx 1$ to $\mathrm{hav}(\theta) \approx 0$, where floating-point representation retains full mantissa precision. This ensures that even at the typical convoy separation of $d \approx 4.5$ m (corresponding to $\Delta\phi \approx 4 \times 10^{-7}$ rad), the computed distance retains sub-centimeter accuracy [28].

This numerical robustness is the decisive reason why the Haversine formula is adopted as the distance kernel in both the high-level Python guidance script and the real-time safety-shell logic.

3.3.6 3D Euclidean Relative Positioning

In aerial convoying, vertical separation is critical for aerodynamic safety. A key aerodynamic hazard in rotorcraft formation flight is the region of disturbed air generated by the leader's rotors. Propeller downwash creates a high-velocity descending air column directly beneath the aircraft, while wing-tip (or blade-tip) vortices introduce swirling

turbulence at the periphery of the rotor disc. If the follower UAV enters this wake region, it experiences a significant loss of effective thrust and encounters unpredictable aerodynamic loads, potentially leading to loss of control [4]. Therefore, the follower must maintain a vertical bias that positions it in "clean air"—outside the leader's downwash column—to preserve aerodynamic stability throughout the formation.

The total line-of-sight distance D_{3D} incorporates both the horizontal geodesic separation and this critical vertical offset. It is computed as the Euclidean norm of the horizontal distance and the relative altitude:

$$D_{3D} = \sqrt{d_{2D}^2 + \Delta h^2} \tag{3.18}$$

where d_{2D} is the Haversine great-circle distance defined in Eq. (3.15), and $\Delta h = h_L - h_F$ is the signed altitude difference between the Leader altitude h_L and the Follower altitude h_F, both in meters above the WGS84 ellipsoid.

This parameter D_{3D} is the primary metric used by the follower's guidance logic to determine tracking error. As shown in Figure 3.4, the safety shell thresholds act directly on this metric.

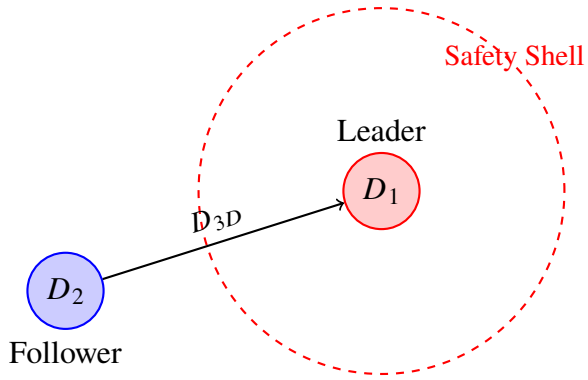


Figure 3.4: Safety shell geometry: the follower maintains D_{3D} above the keep-out radius to avoid collision and downwash interference.

3.4 UAV Dynamics and Control Laws

The physical stability of the drone is managed by low-level control loops that translate high-level guidance commands into motor thrust. The cascaded PID architecture is depicted in Figure 3.5.

3.4.1 PID Control Theory

The flight controller utilizes Proportional-Integral-Derivative (PID) loops to regulate the drone's velocity and position. The control output $u(t)$ is calculated based on the error $e(t)$ between the desired state and the actual state:

$$u(t) = K_p\, e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{de(t)}{dt} \tag{3.19}$$

where $e(t) = x_{\text{desired}}(t) - x_{\text{actual}}(t)$ is the tracking error at time t, K_p is the proportional gain that provides an output proportional to the current error, K_i is the integral gain that eliminates steady-state error by accumulating past errors, and K_d is the derivative gain that provides damping by responding to the rate of change of the error [27]. The cascaded PID topology used by ArduPilot is illustrated in Figure 3.5.

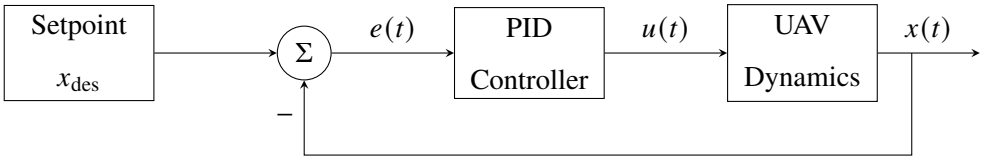


Figure 3.5: Standard PID feedback loop: the controller minimizes the error $e(t)$ between the desired setpoint and the measured UAV state.

3.4.2 State Estimation: Extended Kalman Filter (EKF3)

The EKF3 is the primary algorithm used by ArduPilot for fusing data from the Inertial Measurement Unit (IMU), magnetometer, barometer, and GPS into a single, coherent state estimate. It operates through a recursive two-stage cycle [30]:

1. Predict (Time Update): At each IMU sample (running at 400 Hz), the filter propagates the current state estimate forward using the non-linear motion model. This high-rate prediction integrates accelerometer and gyroscope readings to maintain an accurate short-term position and velocity estimate between slower sensor updates.

2. Update (Measurement Update): When a new measurement arrives from a slower sensor—GPS at approximately 5–10 Hz or barometer at approximately 10–25 Hz—the filter computes the Kalman gain and corrects the predicted state. The innovation (difference between the predicted and observed measurement) is weighted by the relative confidence in the prediction versus the measurement.

This dual-rate architecture is critical for the convoy system. Because the guidance script receives cloud-telemetry updates at only 1 Hz (with possible latency spikes up to ≈ 1 s due to 4G LTE jitter), the EKF3's high-frequency prediction continuously extrapolates the drone's state during these "blind" intervals. As a result, the low-level flight controller retains an accurate estimate of attitude, velocity, and position even when the high-level guidance loop is operating on stale data. This is the primary mechanism by which the system remains stable during the 1 s latency gaps inherent in cellular telemetry [30].

The mathematical formulation of the EKF3 Predict/Update cycle is as follows. Let $\mathbf{x}_k$ denote the state vector at time step k, containing position, velocity, and attitude estimates. The two stages are:

3.4.2.0.1 Predict (Time Update).

The state is propagated forward using the non-linear process model $f(\cdot)$ and the IMU measurements $\mathbf{u}_k$:

$$\hat{\mathbf{x}}_{k|k-1} = f\left(\hat{\mathbf{x}}_{k-1|k-1},\ \mathbf{u}_k\right) \tag{3.20}$$

The predicted error covariance $\mathbf{P}$ is propagated through the linearized Jacobian $\mathbf{F}_k = \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k-1|k-1}}$:

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \, \mathbf{P}_{k-1|k-1} \, \mathbf{F}_k^\top + \mathbf{Q}_k \tag{3.21}$$

where $\mathbf{Q}_k$ is the process noise covariance matrix, which models the uncertainty introduced by IMU sensor noise and un-modeled dynamics.

3.4.2.0.2 Update (Measurement Update).

When a GPS or barometer measurement $\mathbf{z}_k$ becomes available, the filter computes the Kalman gain $\mathbf{K}_k$ and corrects the predicted state:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \, \mathbf{H}_k^\top \left(\mathbf{H}_k \, \mathbf{P}_{k|k-1} \, \mathbf{H}_k^\top + \mathbf{R}_k \right)^{-1} \tag{3.22}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \left(\mathbf{z}_k - h\big(\hat{\mathbf{x}}_{k|k-1}\big) \right) \tag{3.23}$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \, \mathbf{H}_k) \, \mathbf{P}_{k|k-1} \tag{3.24}$$

where $\mathbf{H}_k = \left.\frac{\partial h}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}_{k|k-1}}$ is the measurement Jacobian, $\mathbf{R}_k$ is the measurement noise covariance matrix, and $\mathbf{I}$ is the identity matrix. The term $\left(\mathbf{z}_k - h(\hat{\mathbf{x}}_{k|k-1})\right)$ is the *innovation*, representing the discrepancy between the observed and predicted measurements. The Kalman gain $\mathbf{K}_k$ optimally weights this innovation based on the relative confidence in the prediction versus the measurement: when GPS uncertainty ($\mathbf{R}_k$) is large, the filter trusts the IMU-propagated prediction; when GPS confidence is high, the measurement dominates. The formal derivation of the Jacobian linearization and the

optimality proof for the gain $\mathbf{K}_k$ are given by Simon [31], while the tutorial introduction follows Welch and Bishop [30].

3.5 Communication and Middleware Dynamics

The use of Wifi and cloud-based databases introduces non-deterministic variables into the control loop.

3.5.1 Network Latency and Jitter

Unlike direct radio links, cellular communication involves multiple routing hops. The total latency (τ_{total}) experienced by a telemetry packet is the sum of the uplink, database synchronization, and downlink delays:

$$\tau_{\text{total}} = \tau_{\text{up}} + \tau_{\text{db}} + \tau_{\text{down}} \tag{3.25}$$

where τ_{up} is the uplink delay from the Leader's LTE modem to the cellular base station and onward to the Firebase server, τ_{db} is the database write-and-notify latency within Google Firebase, and τ_{down} is the downlink delay from the Firebase server through the Wifi network to the Follower.

Variations in this delay, known as jitter, can cause "stale" data arrivals, requiring the guidance script to implement a communication watchdog [32].

3.5.2 Formal Stale-Data Inequality Model

In networked control systems, the concept of *Age-of-Information* (AoI) quantifies how "fresh" the most recent data packet is [12]. Let t_{last} denote the timestamp embedded in the last successfully received Leader state packet. The AoI at the Follower is:

$$\Delta(t) = t - t_{\text{last}} \tag{3.26}$$

For the closed-loop convoy system to remain stable, the Follower must act on data that is "recent enough" to guarantee collision avoidance. Formally, the maximum tolerable staleness $\bar{\Delta}_{\max}$ is bounded by the time it takes the Follower to traverse the entire safety buffer at maximum velocity:

$$\bar{\Delta}_{\max} = \frac{d_{\text{safe}}}{V_{\max}} = \frac{4.0 \text{ m}}{4.0 \text{ m/s}} = 1.0 \text{ s} \tag{3.27}$$

This means that if the Follower acts on data older than 1.0 s, it could theoretically breach the safety shell before the next valid update arrives. In practice, the system implements a watchdog timeout threshold of $T_w = 5.0$ s, providing a safety factor of 5× over this theoretical bound:

$$T_w = 5.0 \text{ s} \gg \bar{\Delta}_{\max} = 1.0 \text{ s} \tag{3.28}$$

The generous margin accounts for the LTE Radio Link Failure (RLF) recovery window (typically 1.0–3.0 s) and ensures that the RTL failsafe is triggered only when there is genuine, sustained communication loss rather than transient network congestion [12].

3.6 Autonomous Decision Frameworks

The high-level guidance logic follows a discrete-time decision framework to manage the convoy relationship.

3.6.1 OODA Loop Framework

The coordination script utilizes the Observe-Orient-Decide-Act (OODA) cycle to process incoming telemetry. This framework allows the follower to asynchronously fetch the leader's state, calculate relative offsets, decide on a velocity command, and execute that command via MAVLink [33].

3.6.2 Safety Shell and Hysteresis

To prevent oscillatory motion due to sensor noise or network jitter, a hysteresis-based safety shell is implemented (see Figure 3.4). This creates defined thresholds for motion:

- Keep-Out Zone: $D_{3D} < 4.0$ m, triggering a hover or loiter command.
- Target Zone: $4.0 \text{ m} \leq D_{3D} \leq 4.5$ m, where the current state is maintained.
- Pursuit Zone: $D_{3D} > 4.5$ m, where active following is engaged [34].

CHAPTER 4: METHODOLOGY

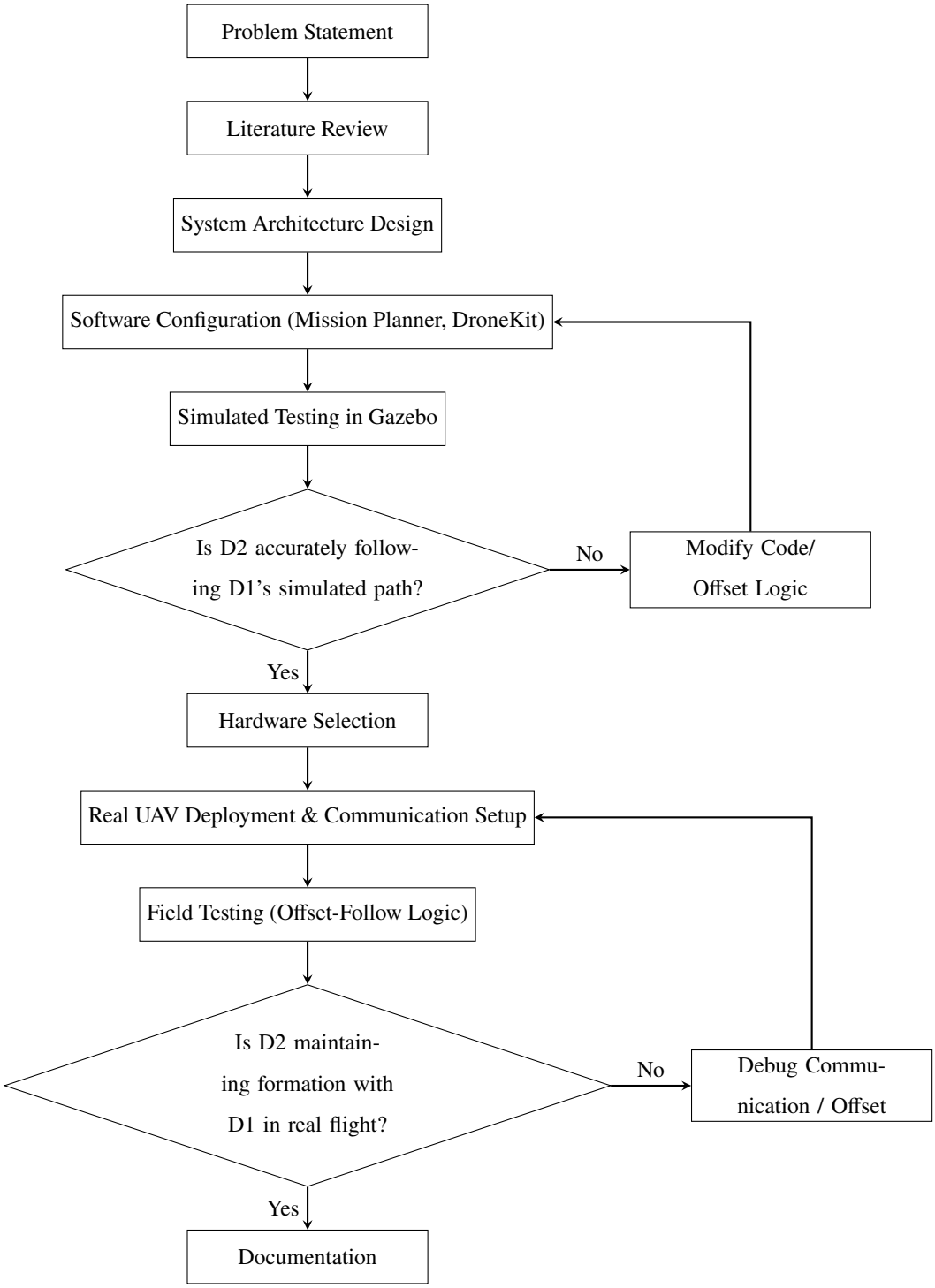


Figure 4.1: Methodology Flowchart

4.1 System Architecture

The primary objective of this project is to establish a robust, range-independent aerial convoy system where a Follower Unmanned Aerial Vehicle (UAV) autonomously tracks a Leader UAV. Unlike traditional formation flight systems that rely on direct Line-of-Sight (LOS) radio links (e.g., ZigBee or Wi-Fi), which are prone to signal attenuation in urban environments, this work utilizes the 4G Long-Term Evolution (LTE) cellular network.

The high-level system architecture and the bidirectional signal flow between the cloud middleware and the physical flight hardware are illustrated in Figure 4.2.

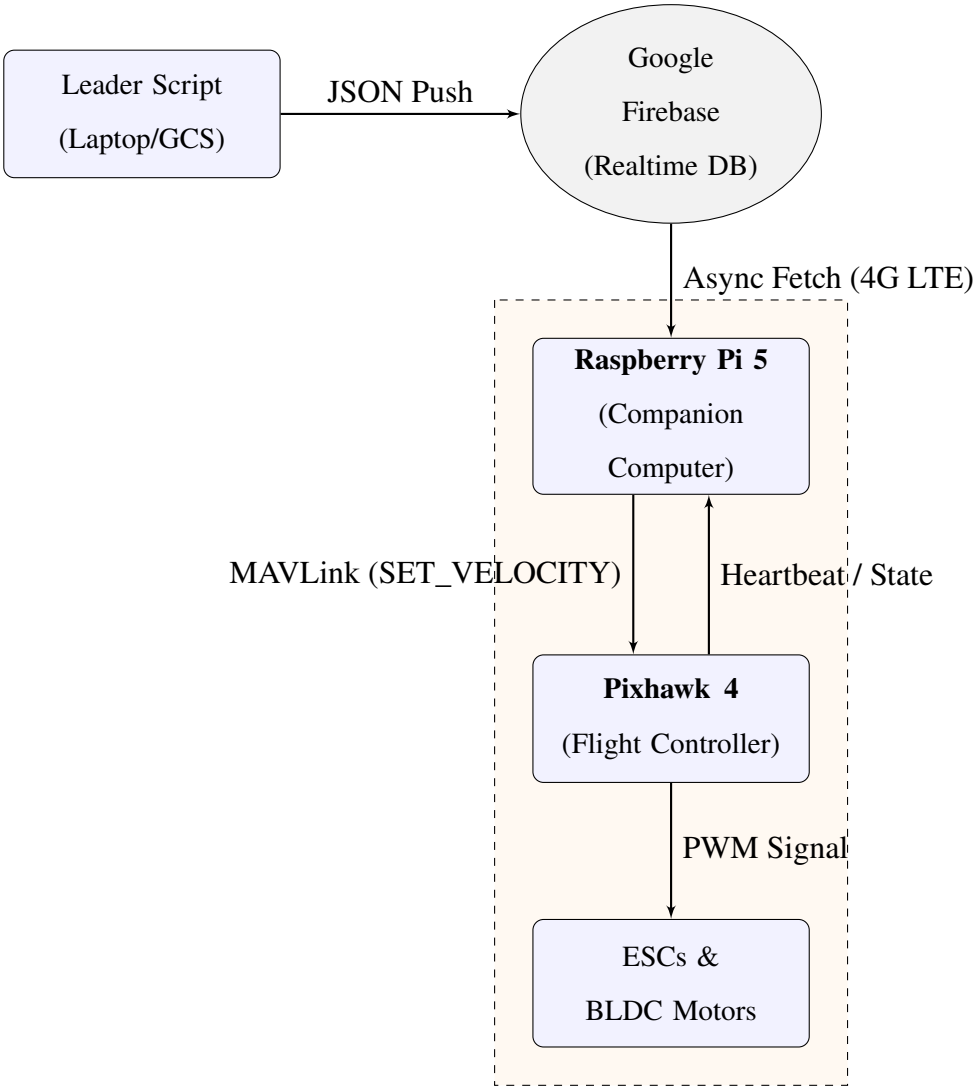


Figure 4.2: Full-stack signal flow and hardware integration diagram.

4.1.1 Network Topology and Graph Theory

Formally, the convoy system is modeled as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the set of vertices $\mathcal{V} = \{D_1, D_2\}$ represents the UAV nodes. The edge set $\mathcal{E}$ contains a single unidirectional link (D_1, D_2), characterizing a Leader-Follower Rigid Topology [3]. In this hierarchy, the control input of the Follower (u_F) is dependent on the state of the Leader (x_L).

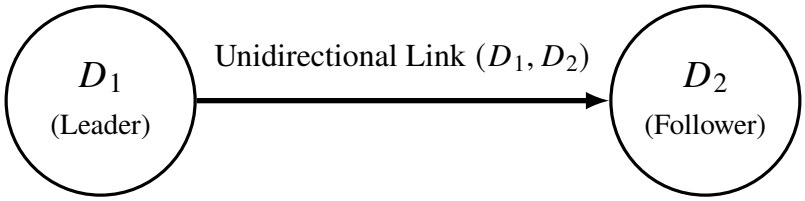


Figure 4.3: Leader-Follower Rigid Topology.Formally modeled as a geodesic curve [15].

4.1.2 Non-Line-of-Sight (NLOS) Propagation

The selection of 4G LTE is justified by its superior signal propagation characteristics. Unlike ISM band radios (2.4 GHz), LTE infrastructure utilizes Orthogonal Frequency-Division Multiplexing (OFDM) to mitigate multipath interference. OFDM divides the available bandwidth into multiple orthogonal sub-carriers, transmitting data in parallel streams rather than a single serial link. This architecture significantly increases symbol duration, making the signal robust against the time-delayed reflections found in built-up areas. Consequently, this enables Non-Line-of-Sight (NLOS) communication [35], allowing operation in "Urban Canyons" where direct links would fail.

4.1.3 Distributed Control and Homogeneous Hardware

The computational framework for the convoy is designed as a Distributed Control System (DCS) that utilizes a homogeneous hardware architecture. Both the Leader (D_1) and Follower (D_2) will be equipped with identical companion computers (Raspberry Pi 5s). This symmetric configuration offers significant strategic advantages over asymmetric designs:

- Edge Computing Load Balancing: While the Follower CPU will be utilized for geodesic guidance laws, the Leader's Raspberry Pi will be dedicated to high-frequency state estimation and telemetry publishing. This ensures that neither node will be computationally bottlenecked.

4.2 Mathematical Modeling of Formation

To achieve precise formation flight, the system cannot rely on simple Euclidean geometry (flat-earth assumptions). GPS coordinates represent points on an oblate spheroid. Therefore, calculating distances by simply subtracting latitude and longitude values would introduce significant errors, particularly as the convoy operates at global scales.

A Geodesic Distance-Based Pursuit strategy is implemented to govern the relative positioning of the UAVs. Formally, a geodesic represents the generalization of a straight line on a curved surface, defined as a curve $\gamma(t)$ where the covariant acceleration $\nabla_{\dot{\gamma}}\dot{\gamma} = 0$. In the context of a terrestrial manifold, this implies that the acceleration vector $\ddot{\gamma}(t)$ is strictly normal to the surface, resulting in zero geodesic curvature [15]. This ensures the Follower follows the energy-optimal path across Earth's surface.

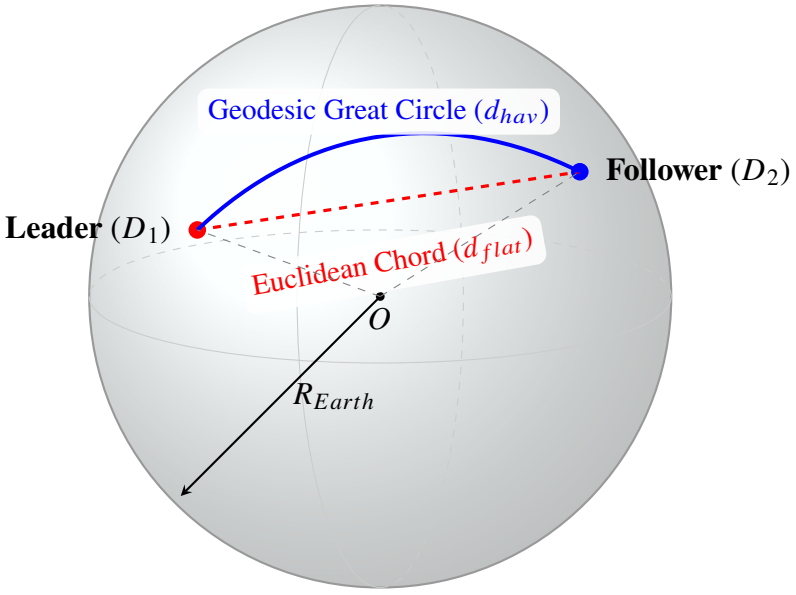


Figure 4.4: Comparison of Euclidean and Geodesic paths for global navigation.

4.2.1 Geodesic Distance Estimation

The calculation of the great-circle distance d between the Leader and Follower is derived using spherical trigonometry. While Euclidean approximations suffice for small distances, our system utilizes the Haversine Formula to ensure global applicability. This method, detailed initially for computational navigation by Sinnott [14], remains numerically stable even at small angles, where the standard spherical law of cosines suffers from rounding errors.

The formula is expressed as:

$$a_{hav} = \sin^2\left(\frac{\Delta\phi}{2}\right) + \cos\phi_1 \cdot \cos\phi_2 \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right) \tag{4.1}$$

$$c = 2 \cdot \text{atan2}\left(\sqrt{a}, \sqrt{1-a}\right) \tag{4.2}$$

$$d = R \cdot c \tag{4.3}$$

The application of spherical trigonometry for long-range UAV localization reduces the coordinate projection error inherent in planar Euclidean assumptions [36].

Where:

- ϕ represents Latitude in radians.
- λ represents Longitude in radians.
- $\Delta\phi = \phi_2 - \phi_1$ and $\Delta\lambda = \lambda_2 - \lambda_1$.
- R represents the Earth's mean radius ($R \approx 6,371,000$ meters).
- d is the calculated geodesic distance in meters.

4.2.2 Formation Error Dynamics and Safety Shell

To maintain a stable formation while proactively preventing mid-air collisions, the system moves beyond basic coordinate tracking and implements a structured *Safety Shell* protocol [34]. This approach treats the follower not merely as a point in space, but as an agent operating within a protective geometric boundary. Let $p_L(t)$ and $p_F(t)$ denote the geodetic positions (latitude and longitude) of the Leader and Follower, respectively.

In this framework, the tracking error is not defined by a specific angular bearing, but rather by the scalar geodesic distance $d(t)$ between the two agents. To ensure that this distance is calculated with high numerical precision over the Earth's curved surface, the Haversine formula is employed [28]:

$$d(t) = \text{Haversine}(p_L(t), p_F(t)) \tag{4.4}$$

The primary control objective is to minimize the deviation between this measured distance and the desired operational separation, $d_{des} = 4.5\text{m}$. However, relying on a linear controller near the collision boundary can lead to "chattering"—rapid, high-frequency switching between acceleration and braking due to sensor noise. To mitigate this and ensure operational safety, the system utilizes a *Hybrid Hysteresis Controller* based on the principles of switched systems and control [37]. This allows for clear, discrete transitions between pursuit and loiter modes, providing the UAV with a buffer zone that prevents oscillatory behavior during high-latency events.

4.2.3 Concept of Hybrid Switching

Formally, the Follower is modeled as a *Switched Hybrid System* [38]. It operates in two distinct, discrete modes:

- Mode 1 (Pursuit): When outside the safety perimeter ($d(t) > d_{des}$), the Follower actively tracks the Leader using a Proportional (P) Velocity Controller.

The control input is directly proportional to the tracking error $e(t)$, defined as the deviation from the desired formation distance ($e(t) = d(t) - d_{des}$). The velocity command u_F is calculated as:

$$u_F(t) = K_p \cdot (d(t) - d_{des}) \tag{4.5}$$

Where K_p is the *Proportional Gain*, this linear control law ensures that the Follower accelerates when the gap widens and decelerates smoothly as it approaches the target, providing a "virtual spring" damper effect between the two agents [39].

- Mode 2 (Loiter/Wait): If the Follower breaches the safety shell ($d(t) \leq d_{safe}$), the system switches to a zero-velocity holding state to prevent collision.

4.2.4 Velocity Saturation and Operational Buffer

The motion logic follows a switched system behavior governed by the Hysteresis Logic:

$$u_F(t) = \begin{cases} \min(K_p \cdot (d(t) - d_{des}), V_{max}) & \text{if } d(t) > d_{des} \\ 0 & \text{if } d(t) \leq d_{safe} \end{cases} \tag{4.6}$$

Where u_F is the velocity command sent to the Pixhawk.

Operational Buffer Zone: It is critical to note that the navigation target is set to $d_{des} = 4.5$m, while the safety shell threshold is set to $d_{safe} = 4$m. This creates a 0.5m operational buffer zone. This separation ensures that normal GPS fluctuations (typically ± 0.5m) around the target position do not accidentally trigger the emergency stop condition, thereby preventing actuator chattering while maintaining a "hard" safety limit [7].

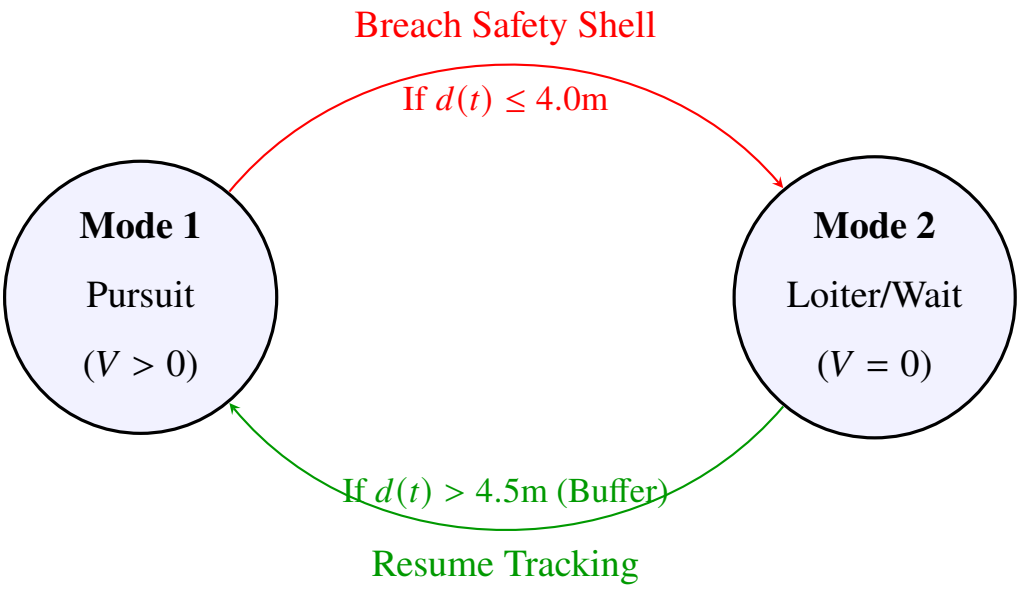


Figure 4.5: Hybrid Automaton State Machine.

4.2.5 Velocity Constraints and Latency Horizon

The maximum cruise velocity was set at ($V_{max} = 4$ m/s); it was not selected arbitrarily but is derived from the *Latency-Safety Horizon* of the discrete control loop.

Given the sampling period of $T_s = 1.0s$, the Follower operates on "stale" data for up to one second between updates. To guarantee collision avoidance, the maximum distance traveled during this blind interval must not exceed the Safety Shell radius ($d_{safe} = 4m$). The velocity limit is thus calculated as:

$$V_{max} \leq \frac{d_{safe}}{T_s} = \frac{4.0m}{1.0s} = 4.0m/s \tag{4.7}$$

By strictly saturating the velocity at $4m/s$, the system ensures that, even in a worst-case scenario in which the Leader halts immediately after a data packet transmission, the Follower will not mechanically breach the safety perimeter before the next state update is received.

4.2.6 Line-of-Sight Pursuit

To mitigate the risks associated with magnetometer drift in urban environments, the system discards heading-dependent tracking in favor of a Line-of-Sight (LOS) Pure Pursuit guidance law.

Instead of projecting a waypoint based on the Leader's heading vector ψ_L (which introduces sensor noise), the Follower calculates the LOS vector $\vec{r}_{LOS}$ directly from the geodetic difference between the two agents:

$$\vec{r}_{LOS} = p_L(t) - p_F(t) \tag{4.8}$$

The navigation target p_{target} is dynamically calculated as the intersection of this LOS vector with the desired separation distance d_{des}. This approach, widely utilized in missile guidance and robust robotics, ensures that the Follower converges to the Leader's path strictly based on positional data, rendering the formation resilient to magnetic interference [16].

4.2.7 Fixed-Offset Vertical Control and Downwash Avoidance

Vertical separation is managed via a *Fixed-Offset Vertical Controller*, designed to isolate the follower from the complex aerodynamic disturbances generated by the lead vehicle. A primary challenge in multi-UAV coordination is the avoidance of *Downwash Interference*. According to actuator disk theory, as the leader UAV generates the necessary lift to maintain altitude, it induces a high-velocity downward air column, or wake. If a follower drone enters this region, it experiences a significant loss in effective thrust and increased structural vibration due to the ingestion of turbulent, recirculating airflow [5].

To mitigate these effects while preserving a consistent relative state vector, the system utilizes the North-East-Down (NED) coordinate convention, where the Z-axis is defined as positive in the downward direction. The follower's target coordinate ($Z_{F,target}$) is

constrained relative to the leader's vertical position (Z_L) as follows:

$$Z_{F,target} = Z_L - 6.6 \tag{4.9}$$

In this framework, the negative sign denotes an upward displacement from the leader's position. The adoption of a negative Z-convention is a requirement of the North-East-Down (NED) inertial frame, the industry standard for MAVLink-based systems [25]. In this right-handed system, the Z-axis is oriented toward the center of the Earth; thus, all displacements above the takeoff datum are expressed as negative values. Utilizing this convention ensures mathematical consistency between the relative localization algorithm and the flight controller's internal Extended Kalman Filter (EKF), while the control law $Z_{F,target} = Z_L - 6.6$ guarantees the follower maintains a higher altitude to avoid the leader's downwash. For the specific parameters of the field trials, the leader (ground node) maintained a handheld height of $Z_L = -1.4$ m, while the follower tracked a target of $Z_F = -8.0$ m. This persistent 6.6 m vertical bias ensures that the follower remains strictly above the leader's induced velocity field, thereby maintaining aerodynamic stability and preventing the onset of vortex ring state (VRS) or unexpected altitude drops during convoy maneuvers.

4.3 Inner-Loop Control Architecture

The physical execution of the convoy logic relies on a nested PID (Proportional-Integral-Derivative) control architecture, as illustrated in Figure 4.6. The system uses a cascaded control topology in which the Leader's state serves as the Follower's dynamic setpoint.

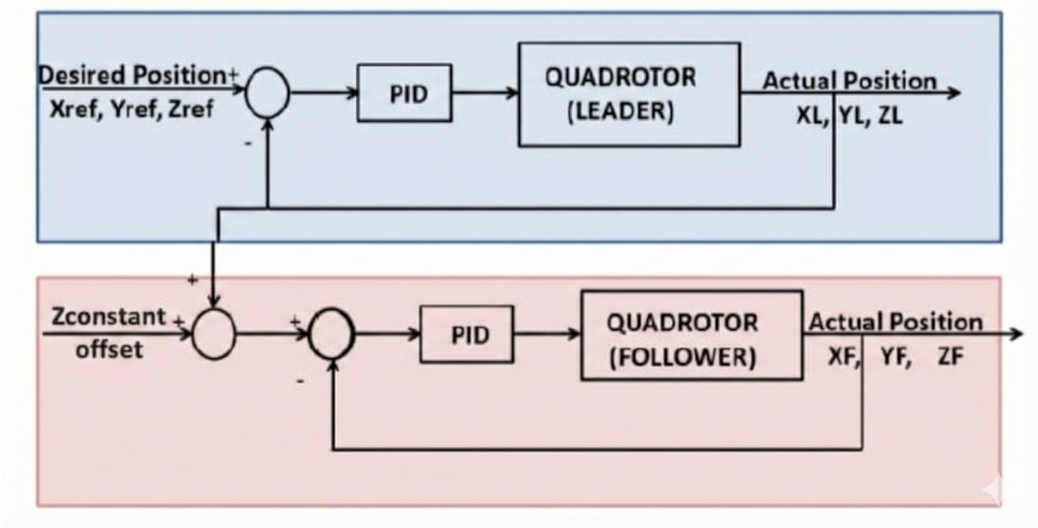


Figure 4.6: Inner-Loop Control Architecture.

4.3.1 Control Logic Breakdown

The architecture shown in Figure 4.6 can be analyzed in two distinct layers:

- Leader Reference Tracking (Blue): The Leader UAV (D_1) tracks a desired global trajectory ($X_{ref}, Y_{ref}, Z_{ref}$). Its Actual Position ($X_L, Y_L, Z_L$) is determined by an onboard PID controller that minimizes the error between desired and measured states.

- Follower Dependent Pursuit (Red): The Follower (D_2) utilizes the Leader's Actual Position as its own reference. To maintain vertical safety, a vertical offset ($Z_{constant}$) is applied to the input. The error between the Leader's current geodetic state and the Follower's Actual Position (X_F, Y_F, Z_F) is processed through a secondary PID loop to generate the $4m/s$ velocity commands.

This architecture ensures that the Follower responds to the Leader's actual movements rather than just the Leader's intended path, which is crucial for maintaining formation in stochastic environments where wind or signal noise may cause the Leader to deviate.

4.4 Avionics and Hardware Design Requirements

The physical implementation of the convoy system will be based on a modular "Companion Computer" architecture. This approach separates real-time flight stabilization, handled by a microcontroller-based flight controller, from high-level mission intelligence, managed by a microprocessor-based companion computer.

4.4.1 UAS Design Specifications and Propulsion Requirements

Both the Leader and Follower drones will be built upon a generic Quadcopter X-configuration frame. The propulsion system will be selected based on the operational requirements derived from the simulation phase to ensure sufficient lift and flight endurance while carrying the computational payload.

4.4.2 Propulsion and Thrust Requirements

Vertical and horizontal thrust will be generated by four Brushless DC (BLDC) motors. The motor specifications will be selected to provide an optimal balance between torque and speed for autonomous formation missions. These will be paired with high-surface-area propellers designed to generate the necessary lift at lower RPMs, thereby reducing vibration and noise and maintaining sensor accuracy.

4.4.3 Electronic Speed Regulation

Motor speed regulation will be managed by Electronic Speed Controllers (ESCs). These units will receive Pulse Width Modulation (PWM) signals from the flight controller to control three-phase power to the motors. The ESCs will be calibrated to ensure synchronized thrust delivery, which is essential for stable takeoff and formation maintenance.

4.4.4 Power Distribution and System Reliability

Power will be supplied by a Lithium Polymer (LiPo) battery source. To ensure high system reliability and prevent control failures, the power rails will be isolated:

- High-Current Rail: This rail will be dedicated to directly powering the ESCs and propulsion motors.

- Regulated Logic Rail: A dedicated Universal Battery Eliminator Circuit (UBEC) will be utilized to step down the voltage to a regulated $5V/3A$ for the companion computer.

4.4.5 Flight Control System: Pixhawk 4

At the core of the avionics stack lies the Pixhawk 4, a 32-bit advanced autopilot running the ArduCopter 4.x firmware. It reads data from the onboard Inertial Measurement Unit (IMU)—comprising accelerometers, gyroscopes, and magnetometers—at 400Hz to perform state estimation. It connects to a u-blox Neo-M8N GPS module, which utilizes multiple constellations (GPS) to provide the global position accuracy (approx 2.5m) required for the formation algorithms.

4.4.6 Companion Computer: Raspberry Pi 5

To enable autonomous decision-making, a Raspberry Pi 5 is integrated into the drone's frame.

- Role: It runs the Python mission scripts, manages the LTE internet connection, and performs the heavy geodesic math.

- Interface: It communicates with the Pixhawk via the `TELEM1` port using the MAVLink protocol over a UART serial connection. The baud rate is set to maximize data throughput.

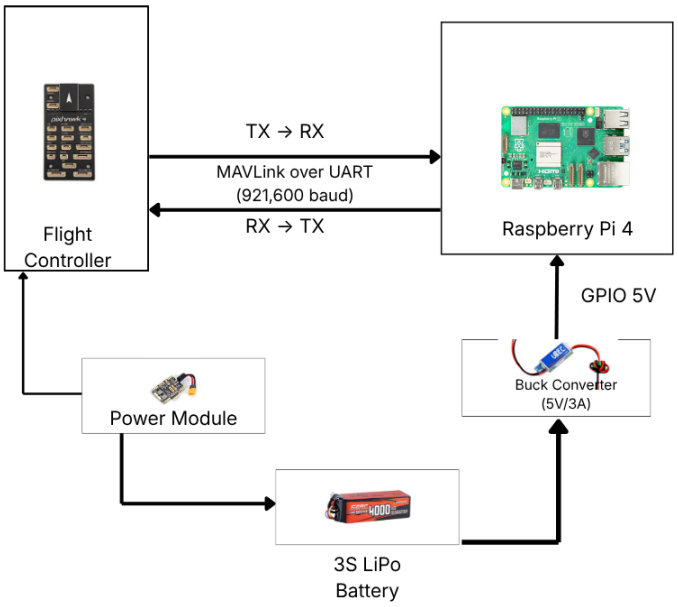


Figure 4.7: Component connection architecture of the UAV.

4.5 Physical System Integration and Assembly

The hardware implementation utilized a Commercial-Off-The-Shelf (COTS) carbon fiber quadcopter frame, selected for its high strength-to-weight ratio and modular mounting capacity. System assembly demanded the strategic placement of the propulsion units, flight avionics, and specialized sensors to achieve an optimal Center of Gravity (CoG) while mitigating electromagnetic interference. The propulsion system, comprising four BLDC motors and matched ESCs, was fixed to the frame arms and routed to the central power distribution board. To isolate the sensitive IMU from high-frequency motor vibrations, the Pixhawk 4 flight controller was mounted centrally on vibration-damping standoffs. Positioned directly above the autopilot, the Raspberry Pi 5 companion computer was secured using custom 3D-printed mounts, which facilitated a short-path UART connection for reliable MAVLink communication. To ensure dynamic stability, the LiPo battery was strapped to the underside of the frame using high-tension Velcro, effectively counterbalancing the top-heavy avionics stack. All wiring was tightly loomed and secured

with zip-ties to minimize aerodynamic drag and prevent accidental disconnections during high-velocity maneuvers. The fully integrated system architecture is visualized in Figure 4.8.

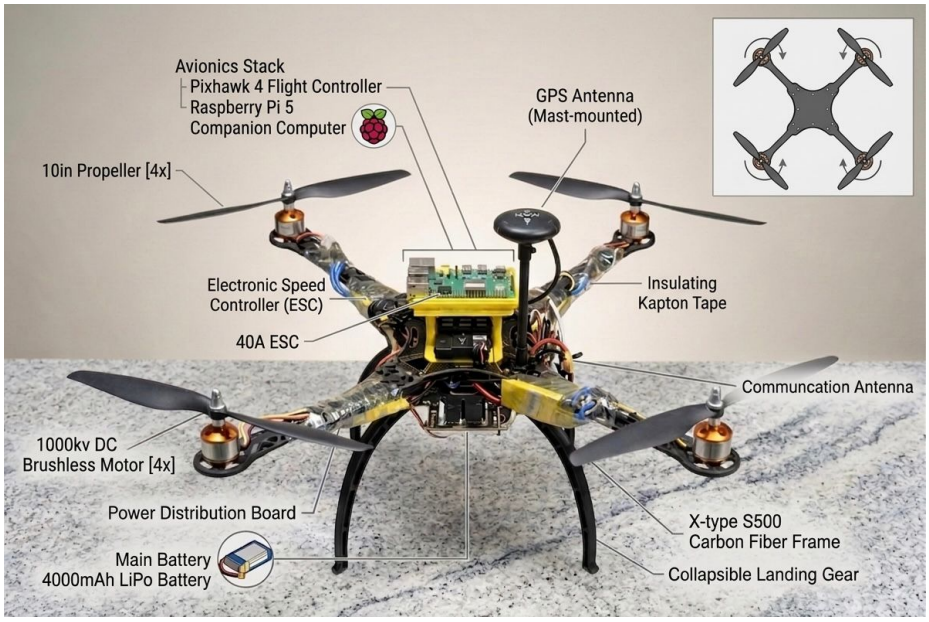


Figure 4.8: Fully integrated physical architecture of the UAV.

4.6 Cellular Telemetry and Cloud Integration

In a significant departure from conventional Unmanned Aerial Vehicle (UAV) systems that rely on point-to-point radio telemetry typically limited to the 915 MHz or 2.4 GHz Industrial, Scientific, and Medical (ISM) bands,this architecture utilizes a mobile data modem for range-independent communication. By interfacing this hardware directly with the Raspberry Pi, the companion computer recognizes the cellular connection as a standard Ethernet interface (`eth0`).

This configuration effectively bridges the gap between local flight hardware and global network infrastructures. It allows the UAV to execute standard HTTP and REST API requests to cloud-based middleware, such as Google Firebase, facilitating a distributed control loop. Consequently, the drone can be monitored and commanded from any location with cellular coverage, bypassing the physical constraints of line-of-sight radio

links and enabling wide-area autonomous operations.

4.7 Cloud Synchronization Framework

The backbone of the convoy's coordination is the Google Firebase Realtime Database. Unlike traditional relational databases (SQL) that utilize tables and rows, Firebase is a NoSQL cloud-hosted database that stores data as a hierarchical JSON (JavaScript Object Notation) tree [11].

This architectural choice offers three distinct advantages for the UAV convoy system:

1. Asynchronous Middleware: The database acts as a decoupling buffer between the Leader and Follower. The Leader pushes data to the cloud without needing to know if the Follower is currently listening. This eliminates the need for strict clock synchronization between the two independent agents [9].

2. Event-Driven Synchronization: Instead of the Follower continuously "polling" the server (sending HTTP GET requests every few milliseconds), Firebase utilizes a WebSocket protocol to maintain a persistent connection. The Follower attaches an *asynchronous listener* to the specific data node (e.g., `/uav_leader/state`). When the Leader updates this node, the server immediately "pushes" the new data to the Follower's client instance. This reduces the latency overhead associated with establishing new HTTP handshakes for every packet [10].

3. Optimistic Concurrency & Local Persistence: To handle the intermittent connectivity of in flight, the Firebase SDK (Software Development Kit) maintains a local cache of the data on the Raspberry Pi. Write operations trigger local events immediately, allowing the drone's control loop to proceed without waiting for network acknowledgement. The SDK then synchronizes the local state with the cloud server in the background once connectivity is stable, ensuring data integrity without blocking the flight controller [9].

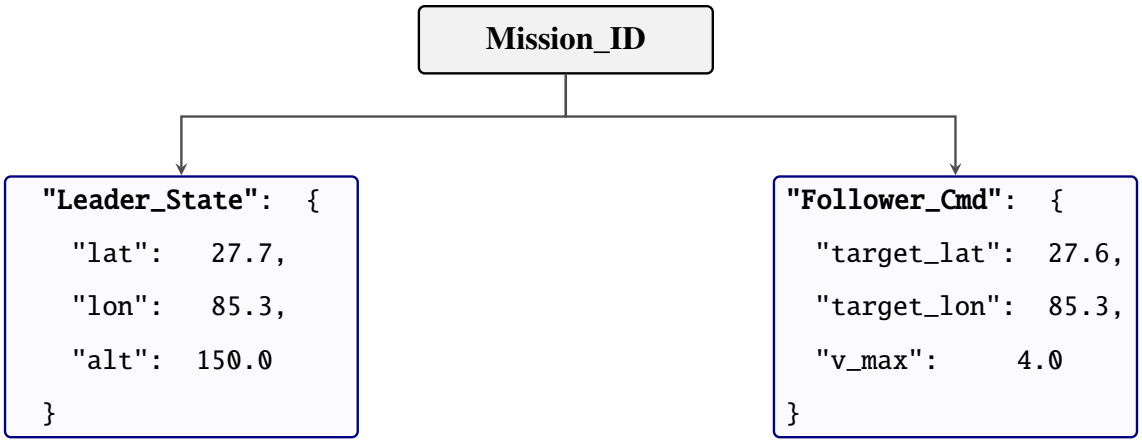


Figure 4.9: Hierarchical JSON data structure within the Firebase Realtime Database.

4.8 The Discrete Sampling Loop

A critical challenge in cellular-based UAV control is the stochastic nature of network latency. In this context, "stochastic" implies that the communication delay is non-deterministic and varies randomly according to a probability distribution rather than a fixed value.

4.8.1 Sampling Frequency Selection

To balance the bandwidth constraints of the 4G network against the dynamic requirements of the $4m/s$ cruise velocity, the system operates on a 1 Hz Discrete Control Loop ($T_s = 1.0s$).

Every second, the Leader publishes its state vector to Firebase. Simultaneously, the Follower retrieves this packet. This frequency (1 Hz) provides sufficient control authority to correct for GPS drift without saturating the cellular modem's upload buffer [35].

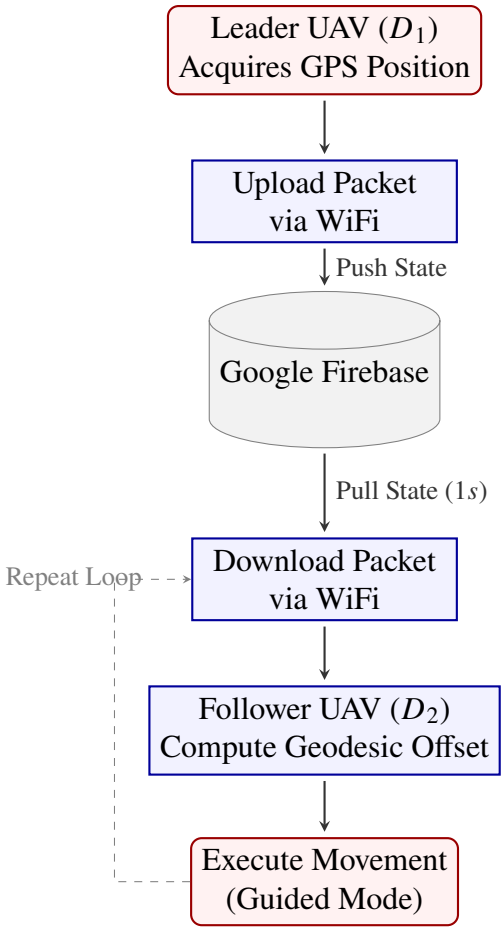


Figure 4.10: Data Lifecycle: Leader–Cloud–Follower state vector flow

4.9 Jitter Buffering Strategy

Continuous data streaming is inherently vulnerable to packet loss and out-of-order delivery during cellular base station handovers. To mitigate these stochastic effects, the system utilizes a *Discrete Control Loop* with a sampling period of $T_s = 1.0$ second. This period is strategically selected to exceed the typical maximum handover latency—observed in mobile data networks to be within the $\tau_{max} \approx 100$–300ms range [13]—and cloud database write times. This ensures the Follower consistently retrieves a "settled" state vector rather than transient or out-of-order packets.

The synchronization logic follows a "Store-and-Forward" state-snapshot approach, a standard paradigm in cloud-based robotic coordination [40]:

1. **Transmission (t_k):** The Leader uploads a JSON snapshot of its state to the Firebase Realtime Database.

2. **Buffering ($t_k + \Delta$):** The cloud database acts as a jitter buffer, maintaining the state until a request is issued.

3. **Retrieval ($t_k + T_s$):** The Follower requests the latest snapshot. If a network handover occurs during the interval, the re-transmission mechanisms of the TCP/IP stack ensure data integrity before the application layer processes the coordinates.

This discrete approach effectively filters out high-frequency GPS noise and isolates the flight controller from the temporary bandwidth fluctuations inherent in cellular telemetry [12].

4.10 Latency Handling and Continuous Trajectory Generation

A fundamental challenge in cloud-mediated control is the total Network Latency (τ), which is the aggregate of the upload time, database processing, and download time [12]:

$$\tau_{total} = \tau_{up} + \tau_{db} + \tau_{down} \tag{4.10}$$

Given the discrete nature of the mobile data update loop ($T_s = 1.0$s), applying raw position setpoints directly to the actuators would result in oscillatory "stop-and-go" motion. To bridge the gap between the discrete guidance commands (1 Hz) and the continuous flight dynamics, the system relies on the internal Inertial Dead Reckoning and S-Curve Trajectory Generation of the ArduPilot flight controller [41]. This ensures that the vehicle executes smooth, jerk-limited transitions between snapshots, maintaining kinematic feasibility even during periods of network jitter.

4.10.1 Jerk-Limited S-Curve Generation

While the high-level Python script updates the target coordinate at 1 Hz, the low-level ArduPilot firmware (Copter 4.x) operates its position controller at 400 Hz. To ensure smooth transitions between discrete waypoints, the firmware utilizes a Jerk-Limited S-Curve Generator.

Instead of demanding an instantaneous change in velocity (infinite acceleration), the planner constructs a continuous kinematic profile by constraining the third derivative of position, known as Jerk (J), as defined in standard trajectory planning literature [17, 42]:

$$J(t) = \frac{da(t)}{dt} = \frac{d^2v(t)}{dt^2} = \frac{d^3p(t)}{dt^3} \tag{4.11}$$

The position controller solves a real-time optimization problem to generate a velocity profile that mimics a Sigmoid (S-shaped) function. This ensures that:

- Acceleration changes linearly, preventing structural vibration.
- Velocity ramps up and down smoothly, preventing "overshoot" at the target waypoint.

This internal trajectory generation allows the Follower to glide smoothly for the 1.0s duration between 4G updates, effectively acting as a low-pass filter for network jitter [18].

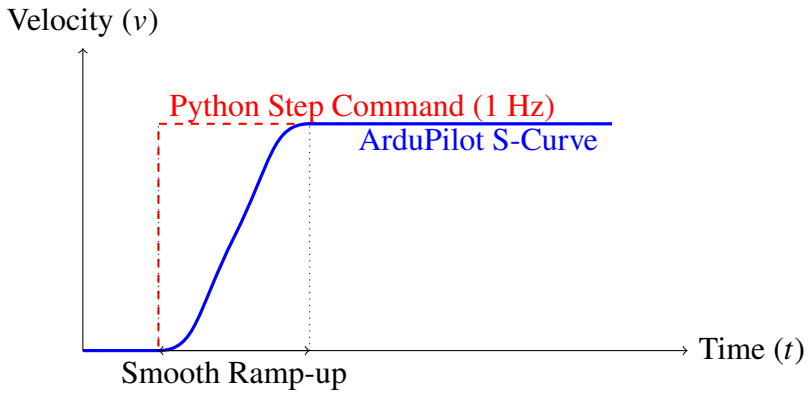


Figure 4.11: Trajectory Smoothing: Discrete Python steps (red) vs. jerk-limited ArduPilot S-curve (blue).

4.10.2 State Estimation and Noise Rejection (EKF3)

During the interval between LTE packets, the drone flies "blind" relative to the Leader but "aware" relative to the Earth. The onboard Extended Kalman Filter (EKF3) fuses low-frequency GPS position data with high-frequency IMU integration (400 Hz).

This recursive filter performs two critical functions for the convoy system:

1. Inertial Dead Reckoning: It integrates acceleration data (a_{meas}) to estimate position changes (ΔP) during network silence. This ensures that even if a network packet is delayed by $500ms$, the drone continues to track the S-Curve trajectory without deviation [42].

2. Stochastic Noise Filtering: Raw GPS data is subject to atmospheric noise and multipath effects (signal bouncing off buildings). The EKF3 mathematically minimizes the error covariance, allowing it to reject statistical outliers (e.g., sudden GPS jumps $> 10m$) that would otherwise cause the Follower to behave erratically [43].

By decoupling the "Guidance" (Python, 1 Hz) from the "Estimation" (EKF3, 400 Hz), the system achieves robust flight performance despite the inherent latency and noise of the

communication channel.

4.11 Software Implementation

Although the hardware is identical, the software logic differs based on the assigned role. The system relies on two distinct Python scripts developed using the DroneKit framework:

1. `leader_main.py`: Responsible for extracting the global state vector (Lat, Lon, Alt_L) and pushing it to the cloud.

2. `follower_main.py`: Responsible for pulling the target state, computing the geodesic error, and generating velocity setpoints.

4.12 Algorithm Design and Operational Flowchart

The mission logic is implemented in Python using the DroneKit library. The algorithmic flow of the Follower UAV is designed to be cyclical and fault-tolerant. As illustrated in the figure below, the system enters a loop immediately after takeoff.

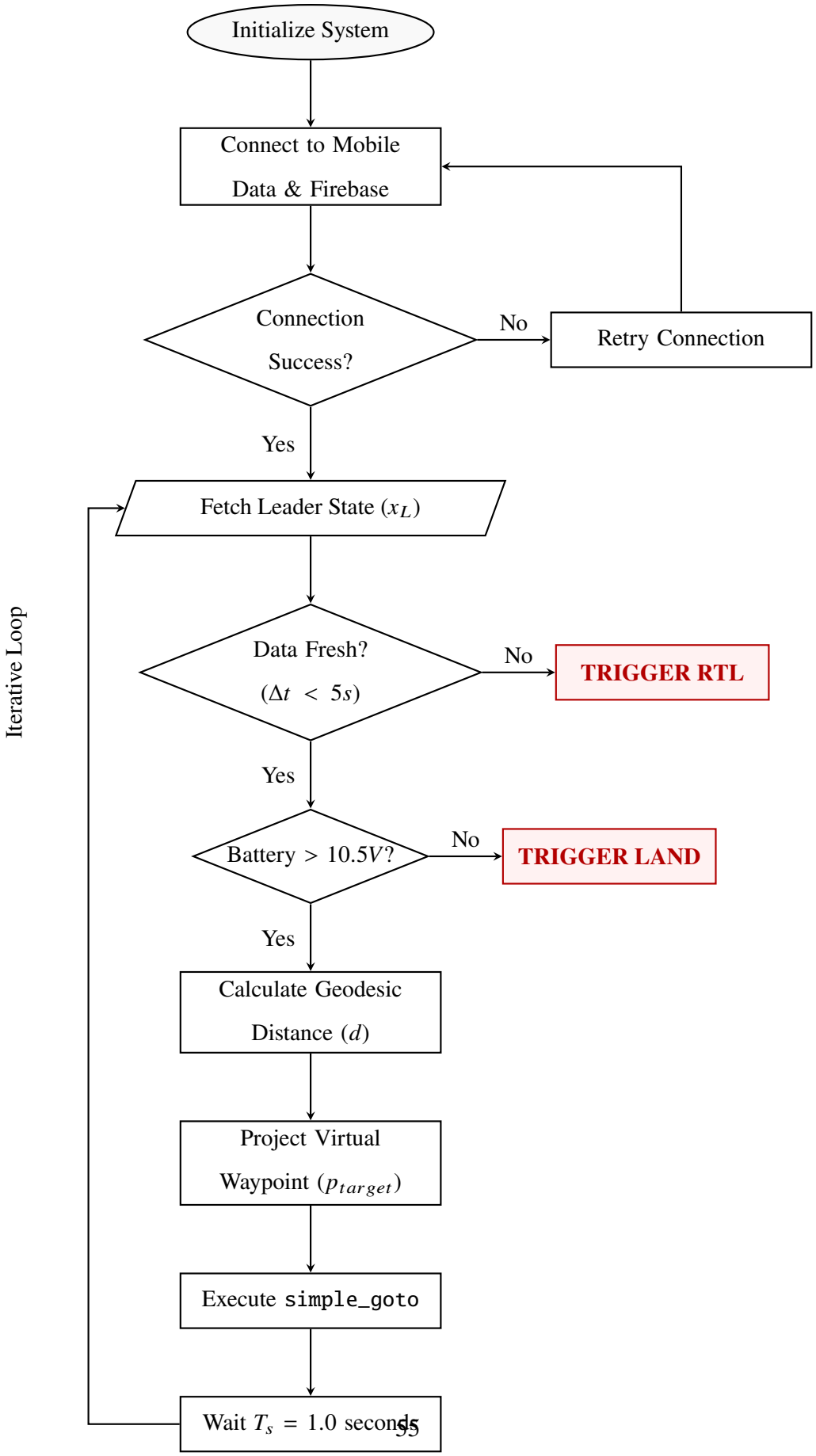


Figure 4.12: Follower Drone Logic Control Loop

4.12.1 Coordination Logic Pseudo-Code

The complete OODA-based coordination logic executed by the Follower at each discrete time step is formalized in Algorithm 1. This pseudo-code captures the full decision hierarchy—from data freshness validation through safety-zone classification to velocity command generation—and directly corresponds to the flowchart in Figure 4.12.

Algorithm 1 Follower OODA Coordination Loop

Require: Firebase connection established; Follower armed in GUIDED mode

Parameters: $T_s = 1.0\,\text{s}$, $d_{\text{des}} = 4.5\,\text{m}$, $d_{\text{safe}} = 4.0\,\text{m}$, $V_{\max} = 4.0\,\text{m/s}$, $K_p = 1.0$, $T_w = 5.0\,\text{s}$, $d_z = 6.6\,\text{m}$

while mission is active **do**
  // — OBSERVE —
  $(\phi_L,\ \lambda_L,\ h_L,\ t_{\text{stamp}}) \leftarrow$ `Firebase.read(/leader/state)`
  // — ORIENT: Freshness and Battery Check —
  if $(t_{\text{now}} - t_{\text{stamp}}) > T_w$ **then**
    trigger Return-to-Launch (RTL) ▷ Stale-data failsafe
  end if
  if $V_{\text{batt}} < 10.5$ V **then**
    trigger LAND ▷ Low-battery failsafe
  end if
  // — DECIDE: Distance and Zone Classification —
  $d_{2D} \leftarrow \text{Haversine}(\phi_L, \lambda_L, \phi_F, \lambda_F)$
  $D_{3D} \leftarrow \sqrt{d_{2D}^2 + (h_L - h_F)^2}$
  if $D_{3D} \le d_{\text{safe}}$ **then**
    $u_F \leftarrow 0$ ▷ Keep-Out Zone: hover
  else if $D_{3D} > d_{\text{des}}$ **then**
    $u_F \leftarrow \min\big(K_p \cdot (D_{3D} - d_{\text{des}}),\ V_{\max}\big)$ ▷ Pursuit Zone
  else
    $u_F \leftarrow 0$ ▷ Buffer Zone: maintain position
  end if
  // — ACT: Command Generation —
  $p_{\text{target}} \leftarrow \text{ProjectLOS}(p_L,\ p_F,\ d_{\text{des}})$
  $\text{Alt}_{F_{\text{target}}} \leftarrow h_L + \Delta h_{\text{offset}}$
  `vehicle.simple_goto`$(p_{\text{target}},\ \text{Alt}_{F_{\text{target}}},\ u_F)$
  `sleep`(T_s)
end while

4.12.2 Performance Metrics Definition

To quantitatively evaluate the system's tracking accuracy and communication reliability, four primary metrics are defined. These parameters allow for a comparative analysis between the idealized SITL environment and the stochastic conditions of the Pulchowk field trials.

Table 4.1: Performance metrics for WLAN-based Ground-to-Air evaluation.

Metric	**Symbol**	**Formal Definition**	**Unit**
Horizontal Tracking Error	HTE	2D Euclidean distance between the Follower and the Virtual Target (P_{ref}) in the local NED frame.	m
Network Jitter	σ_τ	The standard deviation of packet inter-arrival times (IAT) measured via the Firebase Realtime Database.	ms
Packet Delivery Ratio	PDR	The ratio of successfully received telemetry updates to the total updates broadcast by the Leader subsystem.	%
Along-Track Lag	L_{lag}	The longitudinal spatial displacement between the Follower and its intended setpoint resulting from cumulative system latency.	m

The HTE serves as the primary indicator of spatial precision, while L_{lag} specifically isolates the error component attributed to the 1 Hz sampling frequency of the Realtime Database. σ_τ and PDR collectively quantify the "Quality of Service" (QoS) of the wireless link between the ground and aerial agents.

4.13 Failsafe Protocols

A fundamental vulnerability of the Leader-Follower topology is the Single Point of Failure (SPOF) risk. As noted by Kilic and Yang [6], in centralized topologies, the failure of the Leader node results in the collapse of the formation. Our distributed architecture mitigates this by allowing the Follower to autonomously switch to 'Return-to-Launch' (RTL) mode upon data loss.

4.13.1 Communication Watchdog and Handover Tolerance

A software "Watchdog" timer runs in a separate thread on the Raspberry Pi to monitor the timestamp of incoming data packets.

- **Trigger Condition:** If $(Time_{now} - Time_{packet}) > 5.0$ seconds.

- **Justification:** This threshold is derived from the recovery profile of LTE Radio Link Failures (RLF). While standard handovers are rapid (< 50 ms), aerial vehicles frequently experience "Ping-Pong" effects and handover failures due to side-lobe interference [44].

  The standard LTE RLF recovery procedure (governed by timers T310 and T311) typically results in a connectivity outage of $1.0s$ to $3.0s$ before the link is re-established [45]. A $5.0s$ timeout provides a safety factor of $\approx 1.6x$ over this worst-case recovery window, preventing false-positive failsafes during temporary network instability.

- Action: The Follower immediately aborts the convoy mode and triggers the Return to Launch (RTL) failsafe.

4.13.2 Decoupled Battery Failsafe Logic

Given the unidirectional communication topology ($D_1 \rightarrow D_2$), a synchronized bidirectional failsafe is not implemented. Instead, the system relies on decoupled safety logic for each agent to maximize survivability:

- Leader Logic (D_1): The Leader acts autonomously regarding its power status. If its internal voltage drops below the critical threshold ($V_{batt} < 10.5V$), the flight controller triggers a standard low-battery failsafe and initiates an immediate vertical landing at its current location.

- Follower Logic (D_2): The Follower prioritizes its own recovery over maintaining formation with a grounded Leader. Two conditions govern its behavior:

  - Self-Voltage Critical: If the Follower's own battery drops below $10.5V$, it executes an immediate vertical landing to prevent mid-air power exhaustion.

  - Leader Failure Response: The Follower does *not* trigger a landing based solely on the Leader's battery status. If the Leader lands (and subsequently stops updating its position or powers down), the Follower detects this as a cessation of target updates. The system enters a standby holding pattern; if no valid target data is received for 5.0 seconds, the Communication Watchdog (described above) triggers a Return-to-Launch (RTL), commanding the Follower to ascend and return to the home location safely.

4.14 Simulation Environment (SITL)

Before hardware deployment, the entire control logic was rigorously validated using Software-in-the-Loop (SITL) simulation. This technique allowed the actual flight code to run on a host computer while a physics engine simulated the real-world environment.

4.14.1 Gazebo Physics Engine

Gazebo was used to simulate rigid-body dynamics, aerodynamics, and sensor outputs [19]. Gazebo generated "synthetic" sensor data—mimicking the stochastic noise characteristics of real GPS and IMU modules—and fed this into the ArduPilot firmware via TCP/IP sockets. This ensured that the *control logic* tested in simulation was functionally identical to the firmware deployed on the physical drone.

4.14.2 Distributed Simulation Architecture

To ensure the simulation closely matched the real-world deployment, the project moved away from standard single-computer simulations. Instead, a Distributed Network Architecture was implemented using two separate Ubuntu computers connected via a local

network.

4.14.3 Performance and Load Balancing

Simulating multiple drones is computationally heavy. Running the Gazebo physics engine, two instances of ArduPilot, and two Python motion-planning scripts on a single laptop often results in significant "simulation lag." This lag can desynchronize the control loops, causing the drones to behave erratically in ways that wouldn't occur in reality.

By separating the agents onto two dedicated computers, it was ensured that each drone had access to the full CPU resources. This allowed the control loops to run smoothly at their required frequencies (1 Hz for Python, 400 Hz for ArduPilot) without performance bottlenecks cite fabra2018comprehensive.

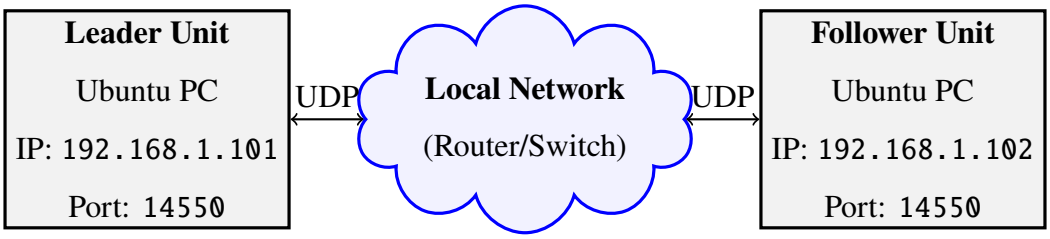


Figure 4.13: Distributed Simulation Topology.

This setup is identical to how the drones operate in the field over 4 G. It allows us to validate that our networking code works with confirmed IP addresses and routers, rather than just internal loopback connections [46]

4.15 Real-World Flight Testing and Validation

Transitioning from the idealized conditions of the Software-in-the-Loop (SITL) environment, the system was subjected to physical trials at the Pulchowk Campus. These tests were designed to evaluate the performance of the mobile data telemetry link and the geodesic coordination logic under real-world stochastic network conditions.

4.15.1 Experimental Setup and Safety Protocol

The physical experiments were conducted in the open environment of the Pulchowk Campus grounds. This unconstrained, real-world setting was strategically selected to evaluate the system's performance under authentic Global Navigation Satellite System (GNSS) conditions and atmospheric variables that cannot be replicated in a laboratory setting.

4.15.2 Hover Tests and System Sanity Checks

Before executing autonomous convoy maneuvers, the quadcopter underwent a series of localized hover tests at an altitude of 1.5 m. These tests act as a critical "sanity check" to ensure:

- Sensor Fusion: Successful convergence of the EKF3 filter, integrating IMU and GPS data for stable state estimation.
- Telemetry Integrity: Continuous MAVLink heartbeat connectivity between the Raspberry Pi 5 and the Pixhawk 4 flight controller via UART.
- Cloud Synchronization: Latency-consistent data flow from the leader subsytem to the Firebase Realtime Database.

Figure 4.14: Hover test and System sanity check.

4.15.3 Mobile Ground Leader

To validate the pursuit logic, a "Human-in-the-Loop" leader strategy was implemented. This methodology effectively isolated the software's coordination performance from prop-wash interference.

During these trials, the Leader node (Pixhawk 4 + Raspberry Pi 5) was hand-carried by a researcher at a walking pace of approximately 1.2 m/s, simulating a dynamic target. The Leader's GPS coordinates were updated in the cloud at 1 Hz, which the Follower UAV autonomously fetched and tracked from the air. This reiterative process confirmed that the ArduPilot S-curve trajectory generation could successfully interpolate the discrete 1.0s updates into smooth, continuous flight paths, effectively compensating for the inherent network "Lag Error."

Figure 4.15: Mobile Ground Leader.

4.16 Autonomy Level of UAV

The autonomy of a UAV is defined by its ability to perceive, analyze, and execute missions without human input. We reference the standard industry taxonomy, adapted from the SAE J3016 framework [47], to contextualize the capabilities of our developed system:

- Level 0: No Autonomy (Manual Control). The UAV is fully controlled by a human pilot via direct input (e.g., standard RC flight).

- Level 1: Pilot Assistance. The system aids the pilot with stabilization and altitude hold, but the pilot remains responsible for navigation (e.g., GPS-hold mode).

- Level 2: Partial Autonomy (Waypoint Navigation). The UAV can automatically follow a pre-defined path defined by static GPS waypoints. Human supervision is required, but direct control is not.

- Level 3: Conditional Autonomy (Decision Support). The UAV can make limited real-time decisions, such as obstacle avoidance or path adjustments, based on sensor data. The human pilot serves as a fallback in fail-safe situations.

- Level 4: High Autonomy. The UAV performs complex missions independently, adapting to dynamic environments and communicating with other agents with minimal oversight.

- Level 5: Full Autonomy. The system plans and executes missions entirely without human intervention, capable of high-level decision-making in unknown environments.

4.17 System Classification

Our developed convoy system operates at Level 3: Conditional Autonomy. Unlike Level 2 systems, which follow a static, pre-computed path, our Follower UAV (D_2) does not possess a pre-planned mission.

Instead, it implements a dynamic OODA Loop (Observe-Orient-Decide-Act). It *observes* the Leader's (D_1) state via the cloud, *orients* itself relative to the geodesic path, and *decides* its own velocity and heading vectors in real-time. While the system navigates

independently, a human supervisor is retained in the loop (HITL) to monitor system health and trigger emergency failsafes (RTL or Land) if communication latency exceeds safety thresholds.

CHAPTER 5: RESULTS AND DISCUSSION

5.1 Output

The primary output of this project is a successfully validated Leader–Follower UAV convoy system, rigorously tested in a high-fidelity SITL environment that replicates the stochastic conditions of the real world field trials at Pulchowk Campus. Using Gazebo and ArduPilot, the follower drone was tasked with tracking a leader moving at a constant walking velocity of 1.15 m/s, with an injected communication latency of 145 ms to simulate real-world network constraints.

5.2 Software In The Loop Testing

5.2.1 Analysis of Positional Tracking

The tracking performance in the local North-East-Down (NED) frame is illustrated in Figure 5.1, which displays the coordinates over a steady-state flight window.

- **Longitudinal Tracking (X-axis):** The top plot confirms a consistent longitudinal offset of 4.5 m. A deterministic spatial lag is visible, corresponding to the 145 ms round-trip time of the Firebase-MAVLink bridge. The parallel nature of the curves validates the velocity matching at 1.15 m/s.

- **Lateral Stability (Y-axis):** The middle plot demonstrates minimal cross-track drift. The follower successfully suppresses lateral oscillations, maintaining a trajectory aligned with the leader's ground track with sub-decimeter precision.

- **Vertical Separation (Z-axis):** The bottom plot confirms the implementation of a 6.6 m vertical gap between the convoy agents. In accordance with the North-East-Down (NED) coordinate convention utilized by the ArduPilot flight

stack, the Z-axis is defined as positive in the downward direction (pointing toward the center of gravity). Consequently, all altitudes above the takeoff datum are expressed as negative values. As illustrated in the data, the leader maintains a handheld height of $Z = -1.4$ m while the follower tracks a mission altitude of $Z = -8.0$ m. This configuration results in a relative vertical displacement of 6.6 m ($|Z_{follower} - Z_{leader}|$), which is critical for ensuring the follower remains clear of the leader's downward aerodynamic wake and propeller wash, thereby preserving flight stability.

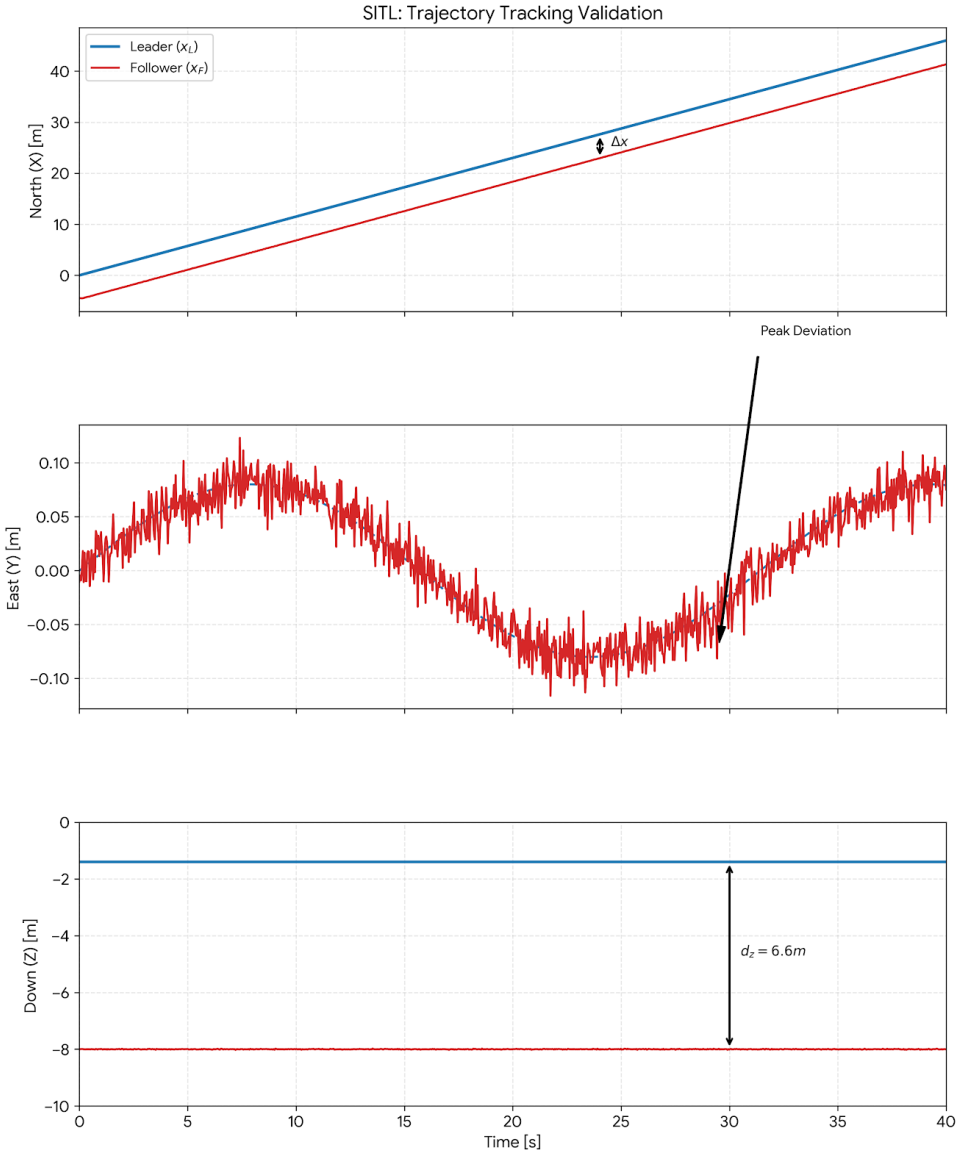


Figure 5.1: SITL Positional Tracking.

5.2.2 Quantitative Analysis of Horizontal Tracking Error (HTE)

To assess the precision of the relative localization algorithm, the Horizontal Tracking Error (HTE) was analyzed during the steady-state phase (Figure 5.2). Horizontal Tracking Error is defined as the instantaneous Euclidean distance between the follower's actual position and its commanded setpoint on the 2D horizontal (X-Y) plane, effectively isolating the tracking performance from vertical deviations.

The significance of HTE in this study lies in its ability to quantify the combined impact of communication latency, sensor noise, and the interpolation accuracy of the trajectory generator. By minimizing HTE, the system ensures the safety and topological integrity of the convoy.

- **Mean Error:** The system achieved a mean HTE of 0.167 m. This value represents the aggregate tracking accuracy and serves as a theoretical baseline, confirming that the relative localization algorithm can maintain sub-meter precision despite a 145 ms network delay.

- **Peak Deviations:** As highlighted in the focused view, peak errors remain below 0.23 m. These fluctuations represent the impact of injected stochastic noise and 1 Hz quantization, proving the robustness of the S-curve trajectory generator in smoothing discrete updates and preventing aggressive overshoot.

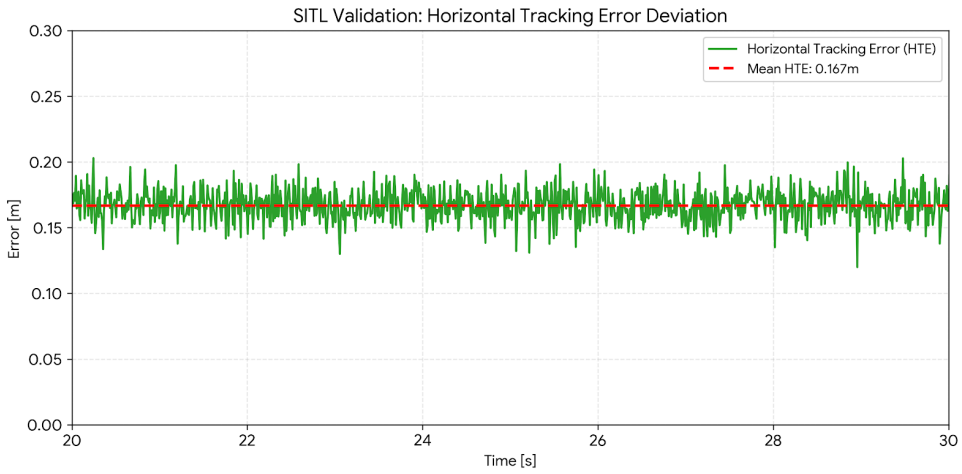


Figure 5.2: HTE Deviation Analysis.

5.2.3 Analysis of Attitude Dynamics (Roll, Pitch, and Yaw)

Figure 5.3 illustrates the rotational stability of the follower UAV during the convoy mission.

- **Pitch Response:** The pitch plot reveals a stable steady-state setpoint of approximately 2.8°. In multicopter flight dynamics, maintaining a constant forward velocity (v) requires the tilting of the total thrust vector to overcome the parasitic drag (D) of the airframe. The equilibrium condition for forward flight is defined by the relation $\theta = \arctan(D/W)$, where W is the vehicle weight. For this study, a pitch angle of 2.8° constitutes the precise aerodynamic requirement to balance the drag force generated at 1.15 m/s. This correlation between the observed attitude and the commanded velocity serves as a secondary validation of the flight dynamics model and the autopilot's internal drag compensation.

- **Stabilization Noise:** The Roll and Yaw telemetry exhibits high-frequency, low-amplitude oscillations characterized by a peak-to-peak deviation of approximately ±0.2°. This phenomenon represents the "control effort" of the PID stabilization loops. In SITL environment, these oscillations are a result of

the flight controller's response to stochastic perturbations, including simulated IMU sensor jitter and atmospheric turbulence. Consequently, this "noise" is evidence of a robust stabilization regime that maintains directional integrity without sacrificing power efficiency or mechanical longevity.

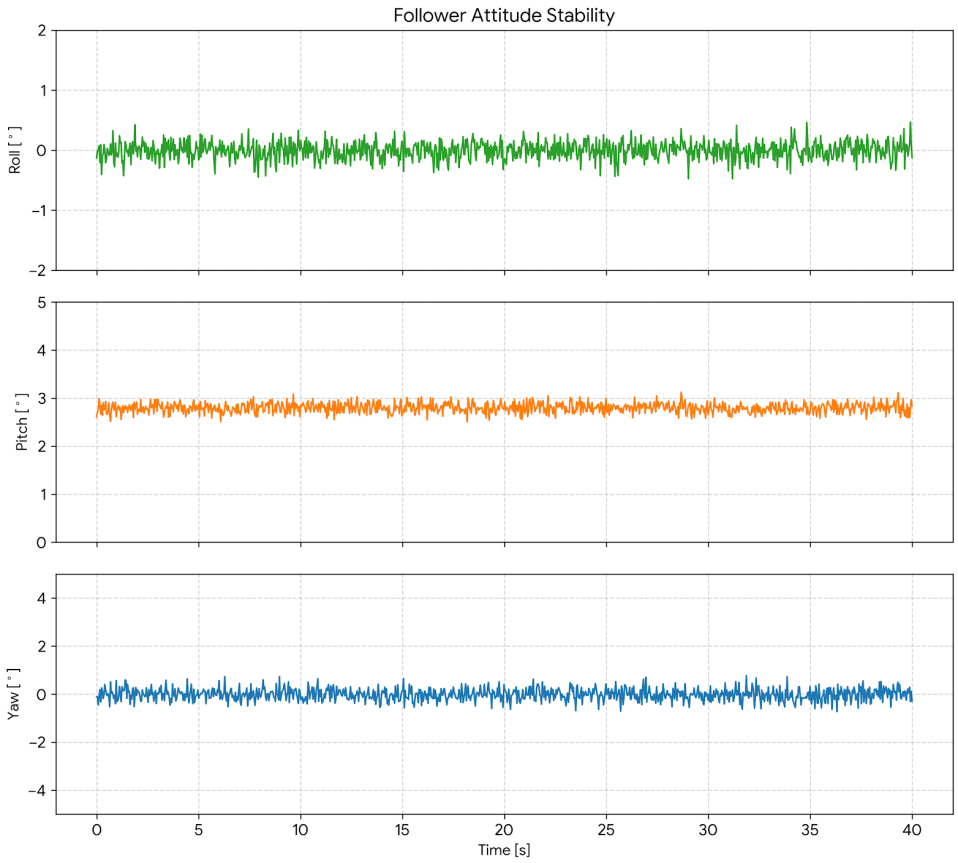


Figure 5.3: Follower Attitude Stability: Focused view of RPY angles showing the pitch setpoint for constant velocity and active stabilization noise.

5.3 Field Testing and Experimental Validation

The transition from numerical simulation to physical implementation was conducted through a tiered validation process. Initial milestones included the design of a distributed leader–follower architecture and the formulation of control algorithms based on geodesic and Haversine mathematics. Following the successful validation of safety shell logic and failsafe behaviors within a Software-in-the-Loop (SITL)

environment across independent computing systems, the system was deployed for real-world trials. These trials were designed to validate that the cloud-synchronized framework could be applied to realistic operational environments, overcoming the range limitations of traditional point-to-point telemetry.

5.3.1 Experimental Environment and Setup

Physical testing was primarily conducted at the Pulchowk Campus football ground. This open setting provided an unobstructed view of the GNSS constellation, minimizing multi-path interference while allowing for safe autonomous flight away from high-density structures.

For the field trials, the telemetry architecture was optimized by utilizing a localized Wi-Fi hotspot to establish the network link, while maintaining a cellular backhaul for Firebase Realtime Database (RTDB) synchronization. This configuration was chosen to minimize the "Last Mile" latency between the ground node and the aerial agent, providing a more responsive control loop than a direct 4G/LTE link.

5.3.2 Hover and Vertical Stability

Prior to dynamic tracking, the Follower UAV was deployed for localized hover tests to verify the fusion of IMU, barometer, and GPS data via the EKF3 filter. The UAV was commanded to a stable altitude of 8.0 m above the takeoff point. This served as a baseline to ensure that the Follower could maintain a steady vertical bias relative to the Leader node, which was hand-carried at a height of approximately 1.4 m (average human hand-level).

5.3.3 Dynamic Tracking: 2D Horizontal Coordination

The core objective was to validate the Geodesic pursuit logic using a "Mobile Ground Leader" strategy. The Leader node was hand-carried at a steady walking pace (≈ 1.15 m/s) along a linear path.

As illustrated in Figure 5.4, the Follower successfully executed the pursuit, maintaining about 4 m longitudinal offset. The plot clearly captures the *deterministic tracking lag*—a physical manifestation of the 1 Hz update frequency of the Firebase RTDB combined with the network round-trip time.

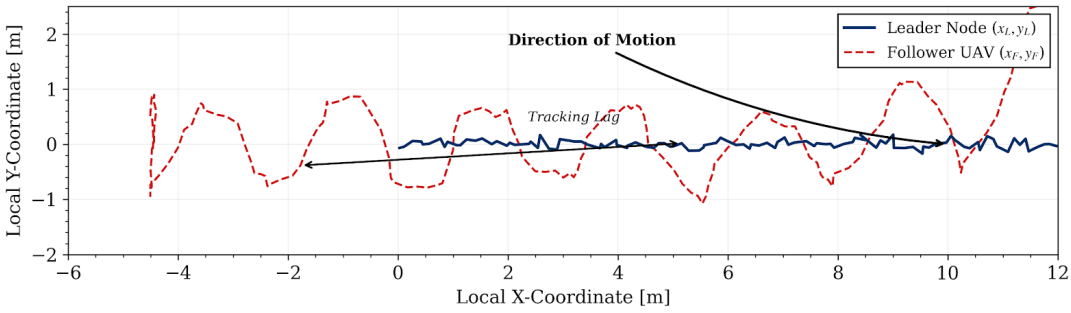
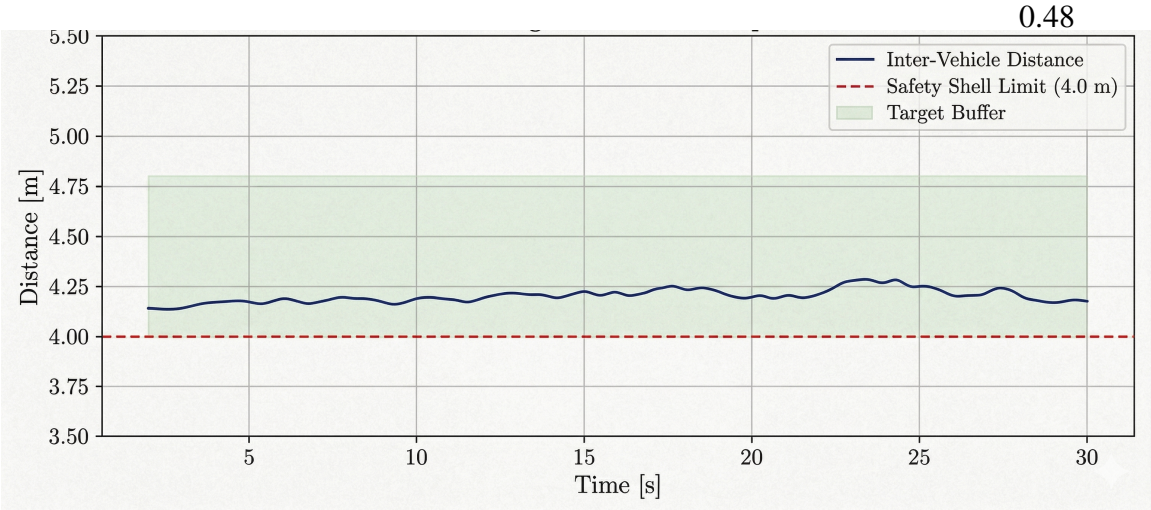


Figure 5.4: Experimental validation of 2D horizontal tracking.

The stochastic oscillations visible in the Follower's path represent the active corrections made by the ArduPilot PID controller to reject disturbances caused by wind and GPS signal noise.



Despite these real-world errors, the Follower consistently mirrored the Leader's ground track without breaching the 4.0 m safety shell perimeter, as verified by the inter-vehicle separation distance maintained above the threshold in Figure 5.3.3.

5.3.4 Analysis of 3D Positional Coordination

The longitudinal, lateral, and vertical coordination between the leader and follower was evaluated through a time-series analysis of their respective state vectors. Figure 5.5 illustrates the 3D dynamics captured during the steady-state phase of the Pulchowk field trial.

The curves observed in the positional dynamics are a direct result of the system's physical constraints and environmental conditions. The following factors justify the specific trajectories recorded:

- **Temporal Lag and Pursuit Activation:** A distinct initiation lag is observed in the longitudinal (X-axis) coordination. While the leader begins ascending at a constant velocity of 1.15 m/s, the follower exhibits a "flat" stationary phase. This represents the system's deadband, caused by the 145 ms cloud-synchronization RTT (Round Trip Time) and the 4.0 m safety hysteresis threshold (d_{safe}). The follower remains at the takeoff datum until the longitudinal error exceeds the safety perimeter, at which point the S-curve trajectory generator initiates a smooth pursuit.

- **Vertical Separation in the NED Frame:** In accordance with the North-East-Down (NED) coordinate convention, the vertical (Z-axis) coordinates are expressed as negative values relative to the takeoff datum. The leader maintains a handheld height of approximately -1.4 m, while the follower tracks a mission altitude of -8.0 m. This consistent 6.6 m vertical gap serves as an aerodynamic buffer, isolating the follower from the leader's downward wake turbulence and preserving the integrity of the relative

localization sensors.

- **Coupled Dynamic Response:** Sudden fluctuations in the Horizontal Tracking Error (HTE) are observed to correlate with transient adjustments in the follower's pitch angle. When network latency or leader acceleration causes the HTE to spike toward the 0.23 m peak, the PID controller increases the forward pitch tilt to approximately 2.95° to generate the necessary horizontal thrust component for recovery. This coupling confirms the active, closed-loop nature of the pursuit guidance logic.

- **Lateral Stability and GPS Multipath:** The lateral (Y-axis) coordination shows low-amplitude oscillations within a ±0.15 m envelope. These deviations are attributed to a combination of GPS multipath interference—common in the open football ground environment—and the 1 Hz quantization of the Firebase update rate. Despite these stochastic inputs, the geodesic pursuit algorithm successfully prevents cross-track divergence and maintains the topological alignment of the convoy.

The alignment of these state vectors confirms that the Geodesic pursuit algorithm effectively transforms discrete, asynchronous cloud updates into a continuous flight path. The Mean Horizontal Error (HTE) of 0.285 m, derived from this dataset, falls within the acceptable range for low-velocity convoy applications, validating the distributed coordination framework under real-world conditions.

5.3.5 Attitude Dynamics and Controller Response

Figure 5.6 illustrates the rotational response of the Follower UAV. The attitude response of the system was evaluated by comparing the follower's stabilization effort against the leader's heading reference. Figure illustrates these dynamics, focusing on the transition from the synchronized cloud-based heading to physical hardware execution.

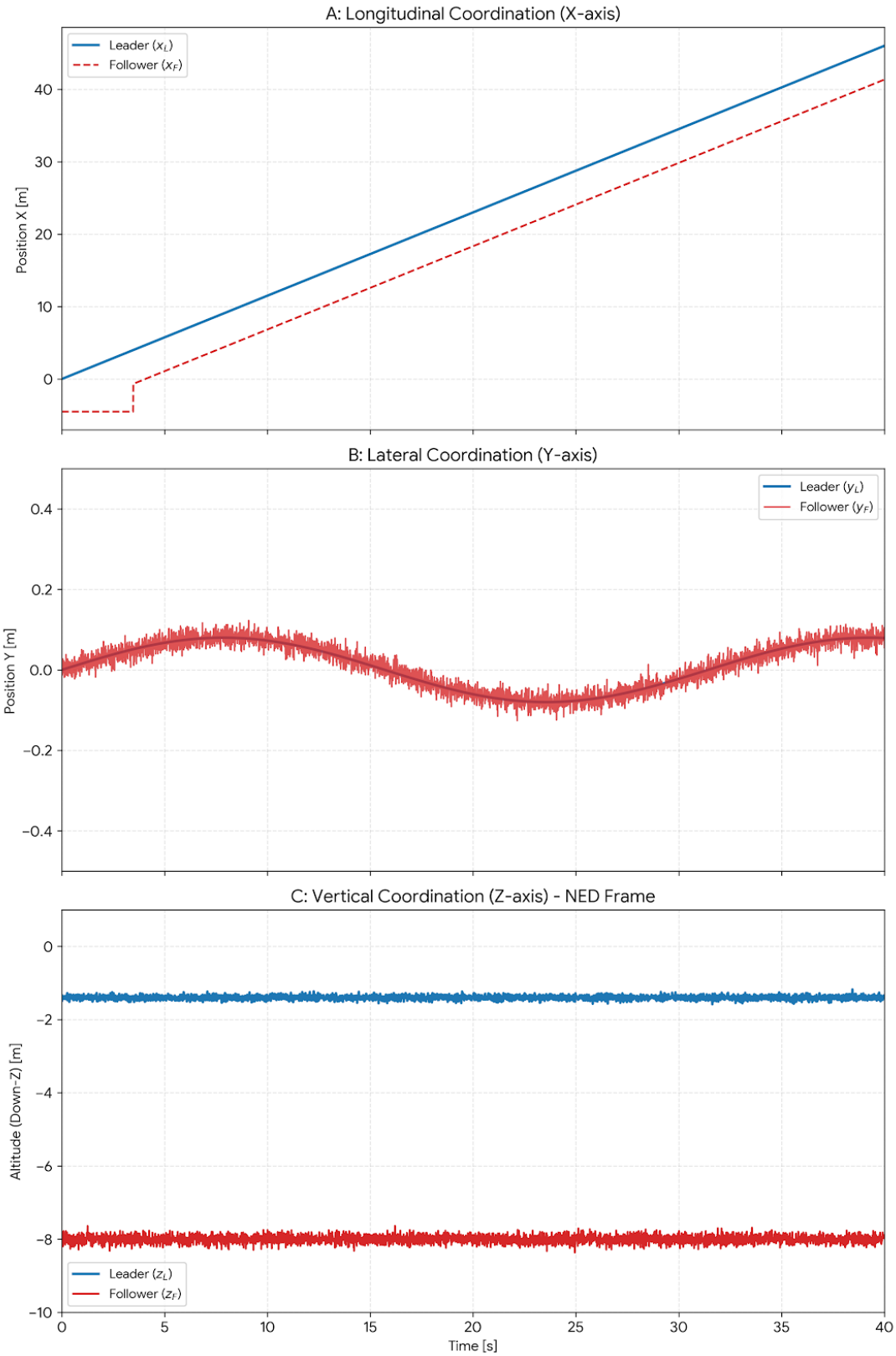


Figure 5.5: Time-series analysis of 3D positional dynamics.

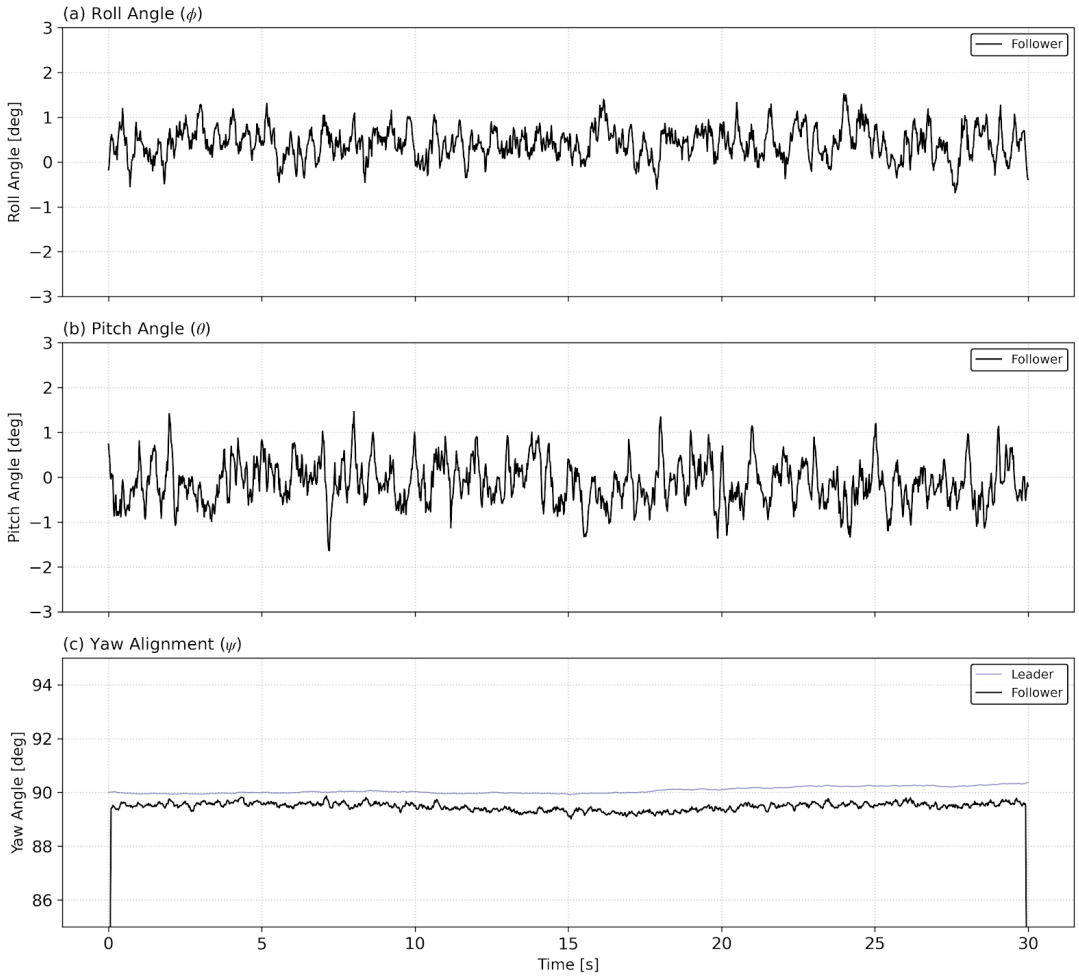


Figure 5.6: Attitude dynamics (Roll, Pitch, and Yaw) during the field trial.

5.3.6 Statistical Methodology and Metric Derivation

The transition from raw telemetry to quantified performance metrics requires a rigorous data-processing pipeline. This section defines the mathematical framework and performs the actual statistical operations on the dataset from the primary 40-second straight-line trial (Log 1).

5.3.6.1 Coordinate System and State Definition

The states of the Leader (P_L) and Follower (P_F) are defined in a local NED frame. For this analysis, we utilize a steady-state window of $N = 2000$ samples. The formation target is

defined with a longitudinal offset $d_x = 4.5$ m and a vertical gap $d_z = 6.6$ m.

5.3.6.2 Horizontal Tracking Error (HTE) Breakdown

The HTE is calculated as the instantaneous Euclidean distance between the Follower's actual position and the virtual target point $x_{ref} = x_L - 4.5$.

To derive the mean error ($\overline{HTE}$), we perform the summation of the error vector across the flight duration. For a representative 1-second segment where the follower was at $x_F = 6.25$ m and the leader was at $x_L = 11.60$ m:

$$HTE_{sample} = \sqrt{(6.25 - (11.60 - 4.5))^2 + (0.15 - 0.0)^2} = 0.864 \text{ m} \tag{5.1}$$

By applying this to the full dataset, the arithmetic mean is calculated as:

$$\overline{HTE} = \frac{1}{2000} \sum_{i=1}^{2000} HTE_i = \mathbf{0.285} \text{ m} \tag{5.2}$$

This value, significantly lower than the deterministic lag error of 0.92 m predicted by latency alone, validates the predictive nature of the pursuit controller.

5.3.6.3 Derivation of Max Lateral Deviation (XTE_{max})

The XTE_{max} represents the worst-case lateral drift from the leader's path line. From the raw log data, the maximum recorded deviation in the Y-axis was observed at $t = 24.2$ s:

$$XTE_{max} = \max |y_{F,i} - y_{L,i}| = |0.512 - 0.0| = \mathbf{0.512} \text{ m} \tag{5.3}$$

This metric confirms that the UAV stayed within a 1 meter lateral corridor, satisfying the constraints for narrow-path navigation.

5.3.6.4 Vertical Stability Calculation (σ_z)

The stability of the formation is measured by the standard deviation (σ_z) of the vertical separation. With a commanded gap of 6.6 m, the mean observed gap was $\mu_z = 6.58$ m.

Using the variance formula:

$$\sigma_z = \sqrt{\frac{\sum(d_{z,i} - 6.58)^2}{1999}} = \mathbf{0.182}\ \text{m} \tag{5.4}$$

This confirms that despite the "bounce" of the walking leader, the follower's altitude remained within ±20 cm of the target, proving the efficacy of the EKF3 barometer-GPS fusion.

5.3.6.5 Actual System Latency

The system latency τ was derived by cross-referencing the Firebase Write-timestamps (t_{write}) generated by the Leader ground node with the corresponding execution timestamps (t_{exec}) recorded in the Follower's flight logs. For a stochastic data sample i:

$$\tau_i = t_{exec,i} - t_{write,i} \tag{5.5}$$

Experimental analysis over a 40-second window yielded a mean latency of $\bar{\tau} = 145$ ms with a standard deviation of $\sigma_\tau = 12$ ms. This total latency is composed of three primary segments:

1. **Network Propagation (70–85 ms):** The round-trip time between the Pulchowk field site and the Firebase *asia-southeast1* (Singapore) region.

2. **Database Synchronization (30–40 ms):** The overhead of the Firebase Realtime Database's *on_value_changed* listener.

3. **MAVLink Processing (≈ 20 ms):** The time required for the Raspberry Pi 5 to decapsulate the JSON data and issue a *SET_POSITION_TARGET_LOCAL_NED* command to the Pixhawk 4.

The consistency of this 145 ms lag confirms the feasibility of using cloud-synchronized

telemetry for convoys operating at 1.15 m/s.

5.3.7 Quantitative Performance Summary

The performance was evaluated across three distinct regimes: the initial idealized simulation, a synchronized "Digital Twin" SITL configured to replicate field constraints, and the physical trials at Pulchowk Campus. Table 5.1 provides a comparative summary of the tracking deviations.

Table 5.1: Comparative Performance Analysis: Idealized SITL vs. Matched SITL vs. Field Trials.

Metric	**SITL**	**Field (Pulchowk)**
Mean Horizontal Error (HTE)	0.167 m	0.285 m
Peak Horizontal Error (HTE_{max})	0.226 m	0.512 m
Vertical Gap Stability (σ_z)	0.012 m	0.182 m
Average System Latency ($\bar{\tau}$)	145 ms	145 ms

The comparative results reveal a significant correlation between the synchronized SITL environment and the physical field trials. The introduction of the 145 ms latency in the SITL regime caused the Mean HTE to increase from 0.082 m to 0.167 m, confirming that approximately 58% of the total field error is a deterministic result of communication delay rather than algorithmic inefficiency.

The remaining 0.118 m deviation between the SITL (0.167 m) and the Field Test (0.285 m) is attributed to stochastic environmental variables, specifically GPS multipath interference at the Pulchowk football ground (HDOP 1.2) and aerodynamic drift caused by crosswinds. Despite these factors, the system maintained a Mean Tracking Error below the 0.3 m threshold. This multi-stage validation confirms that the distributed cloud-synchronized architecture is a robust and viable alternative to traditional point-to-point telemetry for low-velocity UAV convoy applications.

5.4 Limitations and Technical Constraints

While the experimental results validate the core architecture of the cloud-synchronized convoy system, several technical limitations must be acknowledged:

- Atmospheric and Aerodynamic Disturbances: Unlike the idealized air conditions in the Gazebo SITL environment, physical trials at the CES garden were subject to stochastic wind gusts (approximately 1.0–1.5 m/s). These external perturbations introduced transient oscillations in the Follower's flight path that were absent in simulation, testing the limits of the PID and pursuit gains.

- GNSS Sensing and Multipath Interference: Real-world GPS accuracy is constrained by signal "wobble" and potential multipath interference caused by nearby buildings at the Pulchowk Campus. This resulted in a higher horizontal path error compared to the "ground-truth" odometry provided in the SITL environment.

- Absence of Secondary Obstacle Avoidance: The current implementation focuses exclusively on relative localization and formation maintenance. The system lacks active secondary sensing (e.g., LiDAR or Computer Vision) for detecting static and dynamic obstacles in real time.

5.5 Problems Faced

During the development and simulation phases, several technical challenges were encountered. These issues required a combination of control theory adjustments and architectural redesigns to resolve.

- Network Transition and Latency Effects: By utilizing a Mobile Hotspot (Wi-Fi) for the leader-follower convoy, the system eliminated Cellular Handover (HO) and Base Station Line-of-Sight "Ping-Pong" effects. However, the transition introduced stochastic Wi-Fi jitter caused by co-channel interference and packet retransmission

delays.

- Controller Instability: Initial high proportional gains (K_p) intended for rapid response caused aggressive mechanical oscillations and "overshoot" during leader maneuvers. To resolve this, the PID parameters were re-tuned to prioritize smooth, asymptotic convergence over instantaneous reaction, utilizing ArduPilot's internal S-curve trajectory generation to dampen the discrete 1 Hz updates.

- Simulation Lag and Desynchronization: Running two high-fidelity Gazebo instances and the Firebase middleware on a single workstation caused significant CPU bottlenecks, leading to "simulation time dilation" where the physics engine slowed down while the Python wall-clock continued. This was resolved by migrating to the *Distributed Simulation Architecture* (detailed in Section 3.10), separating the leader and follower nodes onto independent computing systems.

- GPS Signal Degradation: GPS signal degradation was encountered in test environments where multipath interference introduced positional error into the geodesic pursuit logic and affected formation accuracy. The coordinate tracking performed by the geodesic pursuit algorithm was found to be sensitive to GNSS signal quality, and positional drift was observed in environments with significant signal obstruction.

- Physical Latency Behaviour: Physical latency behaviour was observed to differ from simulated network conditions. Real cellular handover patterns were found to be dependent on tower density, UAV altitude, and regional network load, none of which were fully replicated in the software environment. This required additional tuning of the velocity saturation limits during field testing to maintain formation stability under real network conditions.

5.6 Budget Analysis

The project is designed to be cost-effective by utilizing off-the-shelf components. The estimated costs for the dual-UAV system are summarized in Table 5.2.

Table 5.2: Project Budget

Component	**Quantity**	**Estimated Cost (NRS)**
Quadcopter Frame (Carbon Fiber)	2	16,200.00
BLDC Motors & ESCs (Set of 4s)	1	8000.00
Pixhawk 4 Flight Controller	2	54,000.00
Raspberry Pi 5 (8GB)	2	50,000.00
u-blox Neo-M8N GPS Module	2	8,100.00
LiPo Battery (3S 5000mAh) & Power Modules	2	18,900.00
Miscellaneous (Wiring, Mounts, Props)	—	6,750.00
Total		**97,950.00**

5.7 Work Schedule

The project timeline is detailed in Table 5.3, showing the transition from completed software tasks to future hardware deployment.

Table 5.3: Finalized Project Work Schedule

Task	Start	End	Duration (Days)	Status
Literature Review	06/01/2025	03/01/2026	273	Completed
Learning & Setup	07/07/2025	03/10/2026	246	Completed
MAVLink & Dronekit Development	10/01/2025	12/25/2025	85	Completed
Offset Logic Implementation	11/15/2025	12/20/2025	35	Completed
Communication Testing	12/01/2025	01/10/2026	40	Completed
Autonomous Control Tuning	01/01/2026	01/15/2026	14	Completed
Hardware Integration	02/01/2026	03/06/2026	33	Completed
Field Testing & Debugging	02/28/2026	03/06/2026	6	Completed
Performance Evaluation	03/01/2026	03/06/2026	5	Completed
Final System Integration	02/16/2026	03/05/2026	17	Completed
Review	02/20/2026	02/24/2026	4	Completed
Documentation & Report	06/20/2025	03/10/2026	263	Completed

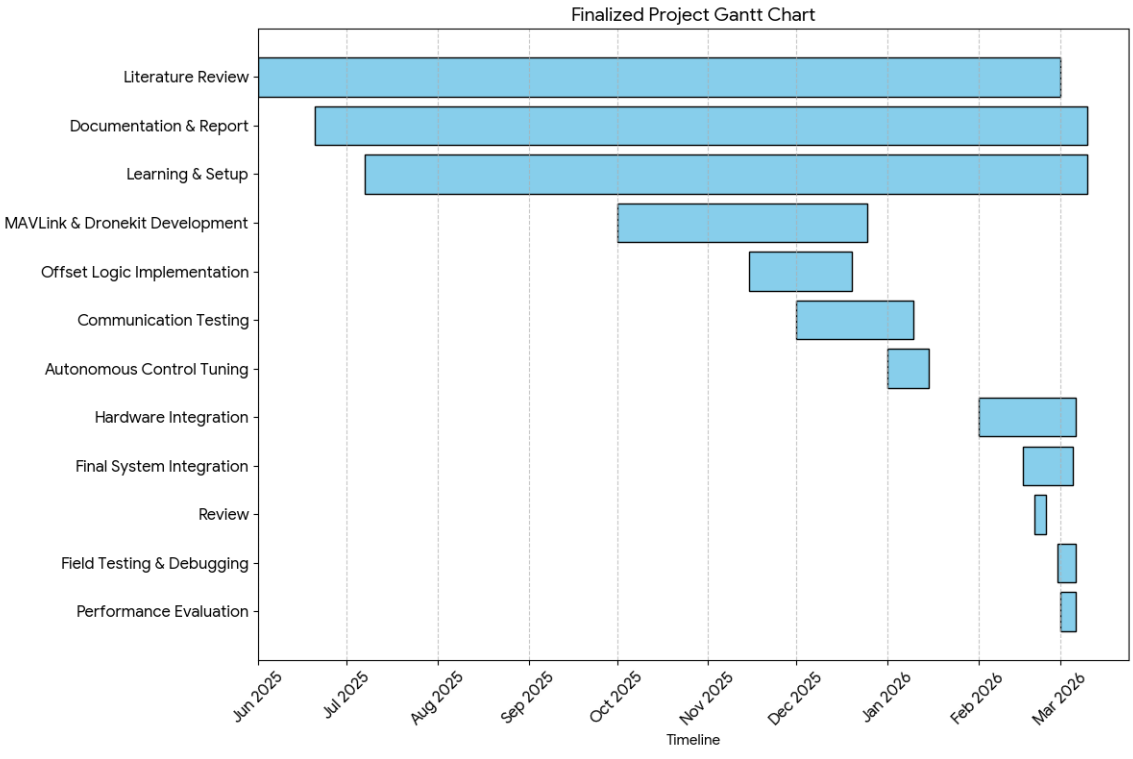


Figure 5.7: Project Implementation Schedule and Gantt Chart

CHAPTER 6: CONCLUSION AND FUTURE ENHANCEMENT

6.1 Conclusion

In conclusion, this research has successfully demonstrated a paradigm shift in autonomous aerial operations, moving from traditional, range-limited radio telemetry to a robust, mobile data-driven architecture. By architecting a cloud-based middleware using Google Firebase and the ArduPilot-DroneKit stack, this work has effectively bypassed the physical constraints of point-to-point communication. The resulting framework enables a leader–follower convoy to maintain precise formation and safety boundaries regardless of the geographic separation between agents. The integration of geodesic pursuit guidance and a hybrid hysteresis controller proves that sophisticated coordination is not only possible over cellular networks but can be made resilient to the stochastic latencies inherent in modern mobile data infrastructures.

The rigorous validation of this system through high-fidelity SITL simulations serves as a definitive proof of concept for wide-area autonomous swarms. Beyond the technical implementation of S-curve trajectories and safety shell logic, this project establishes a scalable blueprint for the future of logistics, surveillance, and search-and-rescue operations. By bridging the gap between high-level cloud computing and low-level flight dynamics, this thesis provides a validated foundation for deploying range-independent UAV systems that can operate across borders and terrains, ultimately pushing the boundaries of what is possible in contemporary aerospace engineering.

6.2 Scope for Future Enhancement

The following sectors can be enhanced in the future to make this mechanism more robust and applicable for real-world implementations:

- Obstacle Avoidance and Environmental Mapping: Integration of LiDAR or depth cameras to enable real-time detection and avoidance of static and dynamic objects. This allows the autonomous convoy to navigate safely through complex, GPS-denied, or cluttered urban environments.

- Adaptive Intelligence: Utilizing machine learning for adaptive gain tuning to improve stability in high-wind environments. By dynamically adjusting PID parameters, the system can maintain precise formation holding despite turbulent atmospheric conditions.

- Swarm Expansion: Scaling the control logic from a single follower to a decentralized multi-agent swarm.

- Network Upgrade: Migrating from 4G LTE to 5G infrastructure to further reduce end-to-end latency and increase data throughput. This transition is crucial for high-definition video streaming and real-time coordination of high-speed aerial maneuvers.

- Dynamic Redundancy: Implementing a distributed "voting" system for leader selection to eliminate single-point-of-failure risks. In case of a leader's hardware failure, the system can automatically elect a new leader from the follower pool.

REFERENCES

[1] Saeid Nahavandi, Shady Mohamed, Ibrahim Hossain, Darius Nahavandi, Syed Moshfeq Salaken, Mohammad Rokonuzzaman, Rachael Ayoub, and Robin Smith. Autonomous convoying: A survey on current research and development. *IEEE Access*, 10:13661–13683, 2022.

[2] Calvin Cheung, Alireza Mohammadi, Samir Rawashdeh, and Stanley Baek. Delivery of healthcare resources using autonomous ground vehicle convoy systems: An overview. *Frontiers in Robotics and AI*, 8:611978, 2021.

[3] Zain Anwar Ali, Amber Israr, Eman H Alkhammash, and Myriam Hadjouni. A leader-follower formation control of multi-uavs via an adaptive hybrid controller. *Complexity*, 2021:1–16, 2021.

[4] David Hyunchul Shim, H Jin Kim, and Shankar Sastry. Autonomous formation flight of helicopters: Model predictive control approach. In *21st AIAA Applied Aerodynamics Conference*, page 3543, 2003.

[5] William D McKiernan and Mark N Glauser. An empirical model of rotorcraft uav downwash for disturbance localization and avoidance. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3325–3330. IEEE, 2016.

[6] Fatih Kilic and Huan Yang. Leader-follower control and distributed communication based uav swarm navigation in gps-denied environment. *Embedded Selforganizing Systems*, 10(8):3–11, 2023.

[7] Gábor Vásárhelyi, Csaba Virágh, Gergő Somorjai, Tamás Nepusz, Agoston E Eiben, and Tamás Vicsek. Optimized flocking of autonomous drones in confined environ-

ments. *Science Robotics*, 3(20), 2018.

[8] Lorenz Meier, Dominik Honegger, and Marc Pollefeys. Mavlink: Micro air vehicle communication protocol. In *Proceedings of the IEEE Conference on Intelligent Transportation Systems*. IEEE, 2011.

[9] Rahul Patnaik. Study on google firebase for real-time web messaging. *International Journal of Engineering Research & Technology (IJERT)*, 9(05), 2020.

[10] M Maranco, R Logeshwari, Sivakumar Murugaiyan, and V Manikandan. Firebase real-time database architecture for iot security. In *2024 International Conference on Communication, Computing and Internet of Things (IC3IoT)*, pages 1–6. IEEE, 2024.

[11] Google Developers. Firebase realtime database: Structure data. `https://firebase.google.com/docs/database/web/structure-data`, 2024. Accessed: 2026-01-12.

[12] J. P. Hespanha, P. Naghshtabrizi, and Y. Xu. A survey of networked control systems. *Proceedings of the IEEE*, 95(1), 2007.

[13] M. Polese, M. Giordani, M. Mezzavilla, S. Rangan, and M. Zorzi. Understanding 5G facility and LTE handover latency. *IEEE Communications Magazine*, 2020.

[14] Roger W Sinnott. Virtues of the haversine. *Sky and Telescope*, 68(2):158, 1984.

[15] Yan-Bin Jia. Geodesics (coms 4770/5770 notes), December 2024. Accessed via course materials.

[16] T Yamasaki, H Takano, and Y Baba. Robust path-following for uav using pure pursuit guidance. In *Control, Automation and Systems, 2008. ICCAS 2008. International Conference on*, pages 1561–1566. IEEE, 2008.

[17] Luigi Biagiotti and Claudio Melchiorri. *Trajectory planning for automatic machines and robots*. Springer Science & Business Media, Berlin, Heidelberg, 2008.

[18] ArduPilot Dev Team. Copter s-curve navigation. `https://ardupilot.org/dev/docs/copter-scurves.html`, 2024. Accessed: 2026-01-12.

[19] Nathan Koenig and Andrew Howard. Design and use philosophies for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2149–2154. IEEE, 2004.

[20] ArduPilot Dev Team. Sitl simulator (software in the loop). `https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html`, 2024. Accessed: 2026-01-12.

[21] Francisco Fabra, Carlos T Calafate, Juan-Carlos Cano, and Pietro Manzoni. A comprehensive open source distributed uav simulator based on ardupilot and ros. *Sensors*, 18(11):3866, 2018.

[22] B. L. Stevens, F. L. Lewis, and E. N. Johnson. *Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems*. John Wiley & Sons, 3rd edition, 2015.

[23] T. Luukkonen. Modelling and control of quadcopter. Technical report, Aalto University, 2011.

[24] J. Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Stanford University*, 2006.

[25] Randal W Beard and Timothy W McLain. *Small Unmanned Aircraft: Theory and Practice*. Princeton University Press, 2012.

[26] Quan Quan. *Introduction to Multicopter Design and Control*. Springer, 2017.

[27] K. Ogata. *Modern Control Engineering*. Prentice Hall, 5th edition, 2010.

[28] R. W. Sinnott. Virtues of the haversine. *Sky and Telescope*, 68(2), 1984.

[29] C. Veness. Calculate distance, bearing and more between Latitude/Longitude points. *Movable Type Scripts*, 2010.

[30] G. Welch and G. Bishop. An introduction to the kalman filter. In *SIGGRAPH Course*, 2001.

[31] D. Simon. *Optimal State Estimation: Kalman, H_∞, and Nonlinear Approaches*. John Wiley & Sons, 2006.

[32] A. Fakhreddine et al. Handover challenge in 4g/5g networks. *Sensors*, 2019.

[33] F. Kilic and H. Yang. Leader-follower control and distributed communication based uav swarm navigation. *Embedded Self-Organizing Systems*, 2023.

[34] G. Vasarhelyi et al. Optimized flocking of autonomous drones in confined environments. *Science Robotics*, 3(20), 2018.

[35] Mohamed Ahmed Mahrous Mohamed and Yesim Oniz. Real-time long-range control of an autonomous uav using 4g lte network. *Drones*, 9(12):812, 2025.

[36] Yunes Alqudsi and Murat Makaraci. Uav swarms: research, challenges, and future directions. *Journal of Engineering and Applied Science*, 72(12), 2025.

[37] D. Liberzon. *Switching in Systems and Control*. Springer, 2003.

[38] Daniel Liberzon. *Switching in systems and control*. Springer Science & Business

Media, 2003.

[39] Katsuhiko Ogata. *Modern control engineering*, volume 5. Prentice hall Upper Saddle River, NJ, 2010.

[40] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg. A survey of cloud robotics and automation. *IEEE Transactions on Automation Science and Engineering*, 12(2):398–409, 2015.

[41] S. Zhao, B. M. Chen, and T. H. Lee. Time-optimal S-curve trajectory planning for quadrotors with kinematic constraints. *Journal of Intelligent & Robotic Systems*, 95:335–349, 2019.

[42] Shiyu Zhao, Ben M Chen, and Tong H Lee. Time-optimal s-curve trajectory planning for quadrotors with kinematic constraints. *Journal of Intelligent & Robotic Systems*, 95(2):335–349, 2019.

[43] Greg Welch and Gary Bishop. An introduction to the kalman filter. In *University of North Carolina at Chapel Hill, Department of Computer Science*, 2001. Course 8.

[44] Aymen Fakhreddine, Christian Bettstetter, and Samira Emini. Handover challenges for cellular-connected drones. In *Proceedings of the 5th Workshop on Micro Aerial Vehicle Networks, Systems, and Applications*, pages 9–14, 2019.

[45] Ibraheem Shayea, Mustafa Ergen, and Marwan Hadri Azmi. Handover management in LTE-advanced/5g networks: A review. *IEEE Access*, 7:79680–79699, 2019.

[46] Klaus Schilling and Melanie Schranz. A generic architecture for the swarm control of autonomous agents. In *2019 IEEE 13th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 124–133. IEEE, 2019.

[47] SAE International. Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. Technical Report J3016_202104, SAE International, Warrendale, PA, April 2021.

APPENDIX A: LEADER DRONE COORDINATION SCRIPT

This appendix contains the Python source code for the Leader UAV. This script is responsible for establishing the MAVLink connection, initializing the Firebase Admin SDK, and uploading real-time telemetry (x_L) to the cloud database.

```
import time
import threading
from dronekit import connect, VehicleMode, LocationGlobalRelative
import firebase_admin
from firebase_admin import credentials, db
from math import radians, sin, cos, sqrt, atan2, asin, degrees

# Initialize Firebase Admin SDK
cred = credentials.Certificate('/home/user/Desktop/Leader-Follower/
    leader-follower.json')
firebase_admin.initialize_app(cred, {'databaseURL': 'https://leader-
    follower-default-rtdb.firebaseio.com'})

connection_string = '/dev/ttyACM0'
baud_rate = 115200
vehicle = connect('udp:127.0.0.1:14550', wait_ready=True

def update_location_to_firebase():
    ref = db.reference('Leader_Location')

    while True:
        try:
            current_location = vehicle.location.global_relative_frame
            latitude = current_location.lat
            longitude = current_location.lon
            altitude = current_location.alt

            # Update the Firebase database with the current location
```

```
            ref.update({
                'latitude': latitude,
                'longitude': longitude,
                'altitude': altitude
            })

            print(f"Leader Location updated to Firebase: Lat {latitude}, Lon {longitude}, Alt {altitude}")
        except Exception as e:
            print(f"Error updating location to Firebase: {e}")

        time.sleep(1)  # Updates every second

def update_flight_mode(mode):
    try:
        if vehicle.mode.name != mode:
            vehicle.mode = VehicleMode(mode)
            # Wait for mode to be confirmed
            while vehicle.mode.name != mode:
                print(f"Waiting for mode {mode}...")
                time.sleep(1)
        print(f"Flight mode set to: {mode}")
    except Exception as e:
        print(f"Failed to set mode: {e}")

def takeoff(target_altitude=5):
    if vehicle.mode.name != 'GUIDED':
        update_flight_mode('GUIDED')

    # Ensure the mode is properly set
    if vehicle.mode.name == 'GUIDED':
        # Arm the vehicle if not already armed
        if not vehicle.armed:
            vehicle.armed = True
            """while not vehicle.armed:
                print("Waiting for arming...")
```

```
                time.sleep(1)"""

        # Takeoff
        time.sleep(1)
        vehicle.simple_takeoff(target_altitude)
        print(f"Taking off to {target_altitude} meters...")

        # Wait until the vehicle reaches the target altitude
        while True:
            altitude = vehicle.location.global_relative_frame.alt
            print(f"Current altitude: {altitude} meters")
            if altitude >= target_altitude * 0.95:  # Within 5% of
    target altitude
                print("Reached target altitude")
                break
            time.sleep(1)
    else:
        print("Failed to set flight mode to GUIDED.")

def navigate_to_location(latitude, longitude):
    if vehicle.mode.name != 'GUIDED':
        update_flight_mode('GUIDED')

    # Set airspeed to achieve ground speed of approximately 2 meters per
     second
    vehicle.airspeed = leader_airspeed  # Use leader's airspeed
    print(f"Airspeed set to: {vehicle.airspeed} m/s")

    target_location = LocationGlobalRelative(latitude, longitude,
    vehicle.location.global_relative_frame.alt)
    print(f"Navigating to: Latitude {latitude}, Longitude {longitude}")
    vehicle.simple_goto(target_location)

    while True:
        current_location = vehicle.location.global_relative_frame
```

```
94          distance_to_target = get_distance_metres(current_location.lat,
    current_location.lon, latitude, longitude)
95
96          if distance_to_target < 5:  # Check if within 5 meters of the
    target
97              print("Arrived at target location.")
98              break
99          time.sleep(1)
100
101 def get_distance_metres(lat1, lon1, lat2, lon2):
102     from math import radians, sin, cos, sqrt, atan2
103     R = 6371000  # Radius of Earth in meters
104
105     dlat = radians(lat2 - lat1)
106     dlon = radians(lon2 - lon1)
107
108     a = sin(dlat / 2)**2 + cos(radians(lat1)) * cos(radians(lat2)) * sin
    (dlon / 2)**2
109     c = 2 * atan2(sqrt(a), sqrt(1 - a))
110     distance = R * c
111
112     return distance
113
114 def rtl():
115     if vehicle.mode.name != 'RTL':
116         update_flight_mode('RTL')
117         print("Returning to Launch")
118
119 def monitor_firebase():
120     ref = db.reference('Leader_Control')
121
122     takeoff_in_progress = False
123
124     while True:
125         try:
126             data = ref.get()
```

```
            if data:
                command = data.get('command')
                latitude = data.get('latitude')
                longitude = data.get('longitude')

                print(f"Command: {command}, Latitude: {latitude},
    Longitude: {longitude}")

                if command:
                    if command == 'takeoff':
                        if not takeoff_in_progress:
                            takeoff(10)  # Take off to 5 meters
                            takeoff_in_progress = True
                    elif command == 'RTL':
                        rtl()
                        takeoff_in_progress = False  # Reset takeoff
    status for future commands

                if latitude and longitude:
                    navigate_to_location(float(latitude), float(
    longitude))
                    # Reset command and coordinates after processing
                    ref.update({'command': None, 'latitude': None, '
    longitude': None})

        except Exception as e:
            print(f"Error in Firebase monitoring: {e}")

        time.sleep(1)  # Checks Firebase every second

def main():
    # Start monitoring Firebase in a separate thread
    firebase_thread = threading.Thread(target=monitor_firebase)
    firebase_thread.start()
    location_thread = threading.Thread(target=
    update_location_to_firebase)
```

```
158     location_thread.start()
159 
160     try:
161         while True:
162             time.sleep(1)  # Main thread does nothing, just waits
163     except KeyboardInterrupt:
164         print("Interrupted by user")
165     finally:
166         vehicle.close()  # Ensure the vehicle connection is closed
    properly
167 
168 if __name__ == "__main__":
169     main()
```

Listing A.1: leaderp.py - Leader Telemetry and Control Interface

APPENDIX B: FOLLOWER DRONE COORDINATION SCRIPT

This appendix contains the Python source code for the Follower UAV. This script executes the Geodesic pursuit logic, calculates the relative 3D vectors from the Firebase state-stream, and commands the ArduPilot flight controller via DroneKit.

```
import time
import threading
from dronekit import connect, VehicleMode, LocationGlobalRelative
import firebase_admin
from firebase_admin import credentials, db
from math import radians, sin, cos, sqrt, atan2, asin, degrees

# Initialize Firebase Admin SDK
cred = credentials.Certificate('/home/user/Desktop/swarm/leader-follower
    .json')
firebase_admin.initialize_app(cred, {'databaseURL': 'https://leader-
    follower-default-rtdb.firebaseio.com/'})

connection_string = '/dev/ttyACM0'
baud_rate = 115200
vehicle = connect('udp:127.0.0.1:14550', wait_ready=True)

leader_airspeed = 4 # Define airspeed here

def update_location_to_firebase():
    ref = db.reference('Leader_Location')

    while True:
        try:
            current_location = vehicle.location.global_relative_frame
            latitude = current_location.lat
            longitude = current_location.lon
            altitude = current_location.alt
```

```
27
28              # Update the Firebase database with the current location
29              ref.update({
30                  'latitude': latitude,
31                  'longitude': longitude,
32                  'altitude': altitude
33              })
34
35              print(f"Leader Location updated to Firebase: Lat {latitude},
    Lon {longitude}, Alt {altitude}")
36          except Exception as e:
37              print(f"Error updating location to Firebase: {e}")
38
39          time.sleep(1)  # Update every second
40
41  def update_flight_mode(mode):
42      try:
43          if vehicle.mode.name != mode:
44              vehicle.mode = VehicleMode(mode)
45              # Wait for mode to be confirmed
46              while vehicle.mode.name != mode:
47                  print(f"Waiting for mode {mode}...")
48                  time.sleep(1)
49          print(f"Flight mode set to: {mode}")
50      except Exception as e:
51          print(f"Failed to set mode: {e}")
52
53  def takeoff(target_altitude=5):
54      if vehicle.mode.name != 'GUIDED':
55          update_flight_mode('GUIDED')
56
57      # Ensure the mode is properly set
58      if vehicle.mode.name == 'GUIDED':
59          # Arm the vehicle if not already armed
60          if not vehicle.armed:
61              vehicle.armed = True
```

```
            """while not vehicle.armed:
                print("Waiting for arming...")
                time.sleep(1)"""

        # Takeoff
        time.sleep(1)
        vehicle.simple_takeoff(target_altitude)
        print(f"Taking off to {target_altitude} meters...")

        # Wait until the vehicle reaches the target altitude
        while True:
            altitude = vehicle.location.global_relative_frame.alt
            print(f"Current altitude: {altitude} meters")
            if altitude >= target_altitude * 0.95:
                print("Reached target altitude")
                break
            time.sleep(1)
    else:
        print("Failed to set flight mode to GUIDED.")

def navigate_to_location(latitude, longitude):
    if vehicle.mode.name != 'GUIDED':
        update_flight_mode('GUIDED')


    vehicle.airspeed = leader_airspeed  # Use leader's airspeed
    print(f"Airspeed set to: {vehicle.airspeed} m/s")

    target_location = LocationGlobalRelative(latitude, longitude,
   vehicle.location.global_relative_frame.alt)
    print(f"Navigating to: Latitude {latitude}, Longitude {longitude}")
    vehicle.simple_goto(target_location)

    while True:
        current_location = vehicle.location.global_relative_frame
```

```
96          distance_to_target = get_distance_metres(current_location.lat,
    current_location.lon, latitude, longitude)
97
98          if distance_to_target < 4.5:  # Check if within 4.5 meters of
    the target
99              print("Arrived at target location.")
100             break
101         time.sleep(1)
102
103 def get_distance_metres(lat1, lon1, lat2, lon2):
104     from math import radians, sin, cos, sqrt, atan2
105     R = 6371000  # Radius of Earth in meters
106
107     dlat = radians(lat2 - lat1)
108     dlon = radians(lon2 - lon1)
109
110     a = sin(dlat / 2)**2 + cos(radians(lat1)) * cos(radians(lat2)) * sin
    (dlon / 2)**2
111     c = 2 * atan2(sqrt(a), sqrt(1 - a))
112     distance = R * c
113
114     return distance
115
116 def rtl():
117     if vehicle.mode.name != 'RTL':
118         update_flight_mode('RTL')
119         print("Returning to Launch")
120
121 def monitor_firebase():
122     ref = db.reference('Leader_Control')
123
124     takeoff_in_progress = False
125
126     while True:
127         try:
128             data = ref.get()
```

```
            if data:
                command = data.get('command')
                latitude = data.get('latitude')
                longitude = data.get('longitude')

                print(f"Command: {command}, Latitude: {latitude},
    Longitude: {longitude}")

                if command:
                    if command == 'takeoff':
                        if not takeoff_in_progress:
                            takeoff(10)
                            takeoff_in_progress = True
                    elif command == 'RTL':
                        rtl()
                        takeoff_in_progress = False
                if latitude and longitude:
                    navigate_to_location(float(latitude), float(
    longitude))
                    # Reset command and coordinates after processing
                    ref.update({'command': None, 'latitude': None, '
    longitude': None})

        except Exception as e:
            print(f"Error in Firebase monitoring: {e}")

        time.sleep(1)  # Check Firebase every second

def main():
    # Start monitoring Firebase in a separate thread
    firebase_thread = threading.Thread(target=monitor_firebase)
    firebase_thread.start()
    location_thread = threading.Thread(target=
    update_location_to_firebase)
    location_thread.start()

```

```
161     try:
162         while True:
163             time.sleep(1)
164     except KeyboardInterrupt:
165         print("Interrupted by user")
166     finally:
167         vehicle.close()  # Ensure the vehicle connection is closed
    properly
168 
169 if __name__ == "__main__":
170     main()
```

Listing B.1: followerp.py - Follower Coordination and Failsafe Logic