Hierarchical Control Architecture for Fixed-Wing Aircraft: A Multi-Level Cascaded Approach

Technical Documentation

Inspired by dRehmFlight (Nicholas Rehm)

October 14, 2025

Abstract

This document presents a comprehensive five-level hierarchical control architecture for fixed-wing aircraft, designed for autonomous flight, reinforcement learning research, and flight control education. The architecture implements a cascaded control structure that separates high-level mission planning from low-level actuation, enabling flexible agent design at multiple abstraction levels. The core design—separating attitude (angle mode) from rate (acro mode) control—is directly inspired by dRehmFlight [1], a widely-used open-source flight controller by Nicholas Rehm. We extend this proven architecture with additional high-level layers (waypoint navigation, HSA control) and provide rigorous mathematical foundations suitable for graduate-level study. The system combines Python flexibility for high-level control with C++ performance for time-critical inner loops, achieving real-time performance suitable for both simulation and hardware deployment.

Contents

1 Introduction

1.1 Motivation and Background

Aircraft control is inherently hierarchical. A pilot does not directly think about aileron deflection angles when commanding a turn; instead, they think in terms of desired bank angles, headings, or waypoints. This natural abstraction hierarchy has been formalized in modern flight control systems through **cascaded control architectures**.

The challenge in autonomous aircraft control is to bridge the gap between high-level mission objectives (e.g., "fly to waypoint at 100m altitude") and low-level actuator commands (e.g., "set aileron to -0.3 radians"). A well-designed control hierarchy:

1. **Separates concerns**: Each level addresses a specific abstraction
2. **Enables modularity**: Levels can be swapped or improved independently
3. **Facilitates learning**: Agents can operate at any level appropriate to their task
4. **Provides robustness**: Outer loops handle disturbances, inner loops ensure tight tracking
5. **Matches hardware realities**: Fast inner loops run on dedicated hardware, slower outer loops on general-purpose processors

1.2 Inspiration from dRehmFlight

This control architecture is fundamentally inspired by **dRehmFlight** [1], an open-source flight controller developed by Nicholas Rehm for small fixed-wing and multirotor aircraft. dRehmFlight pioneered an accessible yet rigorous approach to cascaded flight control, clearly separating:

- **Angle mode (stabilized)**: Outer loop controls Euler angles, outputs rate commands
- **Rate mode (acro)**: Inner loop controls angular rates, outputs surface deflections

This separation—which we adopt as Levels 3 and 4 in our architecture—is the foundation of virtually all modern flight controllers, including Betaflight [2], ArduPilot [3], and PX4 [4]. dRehmFlight's clear pedagogical presentation and well-documented design patterns make it an ideal reference for understanding cascaded control.

Our contribution extends dRehmFlight's core two-loop structure by adding:

- **Level 2 (HSA)**: Heading, speed, altitude control for autonomous flight
- **Level 1 (Waypoint)**: 3D navigation with guidance algorithms
- **Level 5 (Surface)**: Explicit surface control layer for RL research
- **Mathematical rigor**: Graduate-level derivations of control laws
- **Python/C++ hybrid**: Flexibility for research with performance for deployment

While our implementation is original, the conceptual design and control philosophy owe a significant debt to dRehmFlight's proven architecture.

1.3 Document Organization

This document is structured bottom-up, starting with the innermost (fastest) control loop and progressing outward:

- **Section 2**: Overview of 5-level hierarchy
- **Section 3**: Level 4 (Rate control) — the critical inner loop
- **Section 4**: Level 3 (Attitude control) — cascaded outer loop
- **Section 5**: Level 2 (HSA control) — flight state tracking
- **Section 6**: Level 1 (Waypoint navigation) — mission-level guidance
- **Section 7**: Level 5 (Direct surface control) — for advanced research
- **Section 8**: Mathematical foundations (frequency domain, stability)
- **Section 9**: Implementation details (software, real-time)
- **Section 10**: Comparison to industry standards

2 Control Hierarchy Overview

2.1 The Five Control Levels

The architecture consists of five hierarchical control levels, each providing a different abstraction:

Table 1: Five-Level Control Hierarchy

Level	Name	Input	Output
1	Waypoint Navigation	3D position (N, E, D)	HSA command
2	HSA Control	Heading, Speed, Altitude	Attitude angles
3	Attitude Control	Roll, Pitch, Yaw angles	Angular rates
4	Rate Control	Angular rates (p, q, r)	Surface deflections
5	Surface Control	Surface deflections	Actuator commands

2.2 Cascaded Control Flow

The control flow propagates through the hierarchy:

$$\text{Waypoint} \xrightarrow{\text{L1}} \text{HSA} \xrightarrow{\text{L2}} \text{Attitude} \xrightarrow{\text{L3}} \text{Rates} \xrightarrow{\text{L4}} \text{Surfaces} \xrightarrow{\text{L5}} \text{Aircraft} \tag{1}$$

Each level acts as the **outer loop** for the level below. This cascaded structure:

- Provides natural frequency separation (outer loops slower than inner loops)
- Enables independent tuning (tune inner loop first, then outer loops)
- Offers robustness (inner loops reject disturbances quickly)
- Supports bypass (agents can command at any level, skipping higher levels)

2.3 Agent Command Flexibility

A key feature is that **agents can command at ANY level**:

- **Level 1 agent**: Commands waypoints, all 5 levels execute
- **Level 2 agent**: Commands HSA, bypasses Level 1, levels 2-5 execute
- **Level 3 agent**: Commands attitude, bypasses 1-2, levels 3-5 execute
- **Level 4 agent**: Commands rates, bypasses 1-3, levels 4-5 execute
- **Level 5 agent**: Commands surfaces directly, only Level 5 executes

This flexibility enables:

- Curriculum learning (train at simple levels first)
- Task-appropriate abstraction (waypoint for navigation, rates for aerobatics)
- Hybrid architectures (classical at low levels, RL at high levels)

2.4 Frequency Hierarchy

Control levels operate at different frequencies, matching their dynamics:

Table 2: Typical Loop Frequencies

Level	Controller	Frequency	Language
1	Waypoint Guidance	1–10 Hz	Python
2	HSA Control	10–50 Hz	Python
3	Attitude (Angle)	50–100 Hz	Python + C++
4	Rate (Acro)	500–1000 Hz	C++
5	Surface Mixer	500–1000 Hz	C++

The 10:1 frequency ratio between adjacent levels ensures clean separation and prevents coupling.

3 Level 4: Rate Control (Inner Loop)

3.1 Overview and Importance

Level 4—rate control—is the **foundation** of the entire control hierarchy. It is the fastest, tightest loop, running at 500–1000 Hz to track angular rate commands with minimal latency. This is the "acro mode" familiar to RC pilots, where stick inputs directly command roll, pitch, and yaw rates.

Rate control is critical because:

1. All outer loops depend on its tight tracking
2. It directly interfaces with aircraft rotational dynamics
3. It runs at highest frequency → most stringent real-time requirements
4. It provides the fast disturbance rejection necessary for stability

3.2 Aircraft Angular Rate Dynamics

The aircraft's rotational motion is governed by Euler's equations in body-fixed coordinates. For a rigid aircraft with principal-axis inertia:

$$I_{xx}\dot{p} = L - (I_{zz} - I_{yy})qr \tag{2}$$
$$I_{yy}\dot{q} = M - (I_{xx} - I_{zz})pr \tag{3}$$
$$I_{zz}\dot{r} = N - (I_{yy} - I_{xx})pq \tag{4}$$

where:

- p, q, r: Roll, pitch, yaw angular rates (rad/s) in body frame
- L, M, N: Roll, pitch, yaw moments (N·m)
- I_{xx}, I_{yy}, I_{zz}: Principal moments of inertia (kg·m^2)

The cross-product terms (e.g., $(I_{zz} - I_{yy})qr$) are gyroscopic coupling effects. For small aircraft with similar pitch and yaw inertias, these are often negligible, simplifying to:

$$\dot{p} \approx \frac{L}{I_{xx}} \tag{5}$$
$$\dot{q} \approx \frac{M}{I_{yy}} \tag{6}$$
$$\dot{r} \approx \frac{N}{I_{zz}} \tag{7}$$

Linearized rate dynamics: Near trim, moments depend linearly on control surface deflections and damping from angular rates:

$$L = q_\infty Sb(C_{l_{\delta_a}}\delta_a + C_{l_p}\frac{pb}{2V}) \tag{8}$$
$$M = q_\infty Sc(C_{m_{\delta_e}}\delta_e + C_{m_q}\frac{qc}{2V}) \tag{9}$$
$$N = q_\infty Sb(C_{n_{\delta_r}}\delta_r + C_{n_r}\frac{rb}{2V}) \tag{10}$$

where $q_\infty = \frac{1}{2}\rho V^2$ is dynamic pressure, S is wing area, b is wingspan, c is chord, and C terms are aerodynamic derivatives.

Substituting into Eqs. (5)–(7):

$$\dot{p} = a_p\delta_a + b_p p \tag{11}$$
$$\dot{q} = a_q\delta_e + b_q q \tag{12}$$
$$\dot{r} = a_r\delta_r + b_r r \tag{13}$$

where a terms are control effectiveness and b terms are damping (typically $b < 0$ for stability).

These are first-order linear systems, amenable to classical PID control.

3.3 PID Controller Design for Rate Tracking

A PID (Proportional-Integral-Derivative) controller is ideal for rate tracking due to its simplicity, robustness, and well-understood tuning methods.

3.3.1 Continuous PID Formulation

The PID control law for tracking a rate command r_{cmd} is:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{14}$$

where $e(t) = r_{cmd}(t) - r(t)$ is the tracking error.
Proportional term $K_p e(t)$:

- Provides immediate corrective action proportional to error
- Larger $K_p \rightarrow$ faster response, but potential overshoot
- Alone, cannot eliminate steady-state error

Integral term $K_i \int e d\tau$:

- Accumulates error over time, eliminates steady-state offset
- Essential for tracking constant rate commands under disturbances
- Can cause *integrator windup* if unchecked (addressed below)

Derivative term $K_d \frac{de}{dt}$:

- Provides damping, reduces overshoot
- Anticipates future error based on rate of change
- Sensitive to measurement noise (requires filtering)

3.3.2 Discrete PID Implementation

For digital implementation at sample time Δt, the PID becomes:

$$e[k] = r_{cmd}[k] - r[k] \tag{15}$$

$$u[k] = K_p e[k] + K_i \sum_{j=0}^{k} e[j]\Delta t + K_d \frac{e[k] - e[k-1]}{\Delta t} \tag{16}$$

Or in incremental form (more numerically stable):

$$P[k] = K_p e[k] \tag{17}$$

$$I[k] = I[k-1] + K_i e[k]\Delta t \tag{18}$$

$$D[k] = K_d \frac{e[k] - e[k-1]}{\Delta t} \tag{19}$$

$$u[k] = P[k] + I[k] + D[k] \tag{20}$$

3.3.3 Anti-Windup for Integral Term

When control saturates (e.g., u hits ± 1 limits), the integral term can grow unbounded—a phenomenon called **integrator windup**. This causes overshoot and sluggish response when error changes sign.

Clamping anti-windup:

$$I[k] = \text{clamp}\left(I[k-1] + K_i e[k]\Delta t, I_{\min}, I_{\max}\right) \tag{21}$$

Typical limits: $I_{\min} = -10$, $I_{\max} = +10$ (tuned empirically).

3.3.4 Derivative Filtering

Raw derivative amplifies high-frequency noise. A first-order low-pass filter smooths the derivative:

$$D_{\text{filt}}[k] = \alpha D[k] + (1-\alpha) D_{\text{filt}}[k-1] \tag{22}$$

where $\alpha \in (0, 1]$ is the filter coefficient. Typical value: $\alpha = 0.1$ (strong filtering) to 0.5 (moderate).

3.4 Multi-Axis Rate Controller

Roll, pitch, and yaw are controlled by **independent PID loops**. This decoupling is justified because:

1. Aerodynamic cross-coupling is weak at high bandwidth (inner loop)
2. Control surfaces primarily affect their corresponding axis (aileron → roll, elevator → pitch, rudder → yaw)
3. Gyroscopic coupling terms are small for typical aircraft

The multi-axis controller is:

$$\delta_a[k] = \text{PID}_{\text{roll}}(p_{cmd}[k], p[k]) \tag{23}$$

$$\delta_e[k] = \text{PID}_{\text{pitch}}(q_{cmd}[k], q[k]) \tag{24}$$

$$\delta_r[k] = \text{PID}_{\text{yaw}}(r_{cmd}[k], r[k]) \tag{25}$$

where $\delta_a, \delta_e, \delta_r \in [-1, +1]$ are normalized control surface deflections.

3.5 C++ Implementation

Rate controllers run at 500–1000 Hz, requiring efficient implementation. We use C++ for performance:

```cpp
// File: cpp/src/pid_controller.cpp
// PID controller class with anti-windup and derivative filtering

class PIDController {
public:
    float compute(float setpoint, float measurement, float dt) {
        // Error
        error_ = setpoint - measurement;

        // P term
        float p_term = config_.gains.kp * error_;

        // I term with anti-windup
        integral_ += error_ * dt;
        integral_ = clamp(integral_, config_.integral_min, config_.
            integral_max);
        float i_term = config_.gains.ki * integral_;

        // D term with filtering
        float derivative = (error_ - error_prev_) / dt;
        derivative_filtered_ = config_.derivative_filter_alpha *
            derivative +
                               (1.0f - config_.
                                  derivative_filter_alpha) *
                                  derivative_filtered_;
        float d_term = config_.gains.kd * derivative_filtered_;

        // Output
        output_ = p_term + i_term + d_term;
        output_ = clamp(output_, config_.output_min, config_.
            output_max);

        error_prev_ = error_;
        return output_;
    }

    void reset() {
        error_ = 0; error_prev_ = 0; integral_ = 0;
        derivative_filtered_ = 0; output_ = 0;
    }
};
```

This implementation is **original work** (not copied from external sources). It incorporates best practices from control literature while being tailored to our architecture.

3.6 Gain Tuning for Rate Controllers

Tuning PID gains is both art and science. A systematic approach:

Step 1: P-only tuning

- Set $K_i = 0$, $K_d = 0$
- Increase K_p until response is fast with acceptable overshoot ($\sim$20%)
- Note: Steady-state error will remain

Step 2: Add D term

- Increase K_d to reduce overshoot and oscillation
- Typical ratio: $K_d \approx 0.1K_p$ to $0.5K_p$
- If noise is excessive, reduce K_d or increase derivative filtering

Step 3: Add I term

- Slowly increase K_i to eliminate steady-state error
- Typical ratio: $K_i \approx 0.01K_p$ to $0.1K_p$
- Watch for oscillation (too much integral gain)
- Set integral limits to 2–3× typical I term value

For typical RC aircraft:

- Roll: $K_p \approx 0.5$, $K_i \approx 0.05$, $K_d \approx 0.1$
- Pitch: $K_p \approx 0.8$, $K_i \approx 0.08$, $K_d \approx 0.15$
- Yaw: $K_p \approx 0.3$, $K_i \approx 0.03$, $K_d \approx 0.05$

4 Level 3: Attitude Control (Outer Loop)

4.1 Cascaded Architecture

Level 3 forms the **outer loop** of a cascaded control structure, with Level 4 as the inner loop. This is the core of dRehmFlight's design [1]:

$$\boxed{\text{Angle error} \xrightarrow{\text{Level 3 PID}} \text{Rate command} \xrightarrow{\text{Level 4 PID}} \text{Surface deflection}} \tag{26}$$

The separation of angle and rate control provides:

- **Frequency separation**: Angle loop $\sim$50–100 Hz, rate loop $\sim$500–1000 Hz
- **Independent tuning**: Tune fast inner loop first, then slower outer loop
- **Disturbance rejection**: Inner loop rapidly corrects rate disturbances
- **Mode switching**: Can bypass angle control for direct rate mode (acro)

4.2 Euler Angle Representation

Aircraft attitude is parameterized by three Euler angles (3-2-1 yaw-pitch-roll sequence):

$$\phi : \text{Roll angle (rotation about body x-axis)} \tag{27}$$
$$\theta : \text{Pitch angle (rotation about body y-axis)} \tag{28}$$
$$\psi : \text{Yaw angle (rotation about body z-axis)} \tag{29}$$

Advantages:

- Intuitive (directly correspond to aircraft orientation)
- Easy visualization
- Minimal parameterization (3 DOF)

Limitation — Gimbal Lock:
At $\theta = \pm 90$ (vertical pitch), roll and yaw become coupled (loss of one DOF). The kinematic equations become singular.

Mitigation:

$$\theta_{\text{safe}} = \text{clamp}(\theta, -85, +85) \tag{30}$$

For aerobatic flight requiring full 360 pitch, use quaternions instead.

4.3 Euler Angle Kinematics

The relationship between Euler angle rates and body angular rates is:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{31}$$

Inverting (assuming $|\theta| < 85$):

$$p = \dot{\phi} - \sin\theta\dot{\psi} \tag{32}$$
$$q = \cos\phi\dot{\theta} + \sin\phi\cos\theta\dot{\psi} \tag{33}$$
$$r = -\sin\phi\dot{\theta} + \cos\phi\cos\theta\dot{\psi} \tag{34}$$

For small angles ($\phi, \theta \ll 1$):

$$\dot{\phi} \approx p \tag{35}$$
$$\dot{\theta} \approx q \tag{36}$$
$$\dot{\psi} \approx r \tag{37}$$

4.4 Cascaded Attitude Controller Design

The Level 3 controller implements PID on attitude angles, outputting rate commands for Level 4.

Roll controller:

$$p_{cmd}(t) = K_{p,\phi}(\phi_{cmd} - \phi) + K_{i,\phi} \int (\phi_{cmd} - \phi)dt + K_{d,\phi} \frac{d(\phi_{cmd} - \phi)}{dt} \tag{38}$$

Simplifying (assuming $\dot{\phi}_{cmd} \approx 0$ for setpoint tracking):

$$p_{cmd} = K_{p,\phi} e_\phi + K_{i,\phi} \int e_\phi dt - K_{d,\phi} \dot{\phi} \tag{39}$$

where $e_\phi = \phi_{cmd} - \phi$.

Using the small-angle approximation $\dot{\phi} \approx p$:

$$p_{cmd} = K_{p,\phi} e_\phi + K_{i,\phi} \int e_\phi dt - K_{d,\phi} p \tag{40}$$

This is a PD controller on angle error with integral correction, using current rate p as derivative feedback (cleaner than differentiating noisy angle measurements).

Similarly for pitch and yaw:

$$q_{cmd} = K_{p,\theta} e_\theta + K_{i,\theta} \int e_\theta dt - K_{d,\theta} q \tag{41}$$

$$r_{cmd} = K_{p,\psi} e_\psi + K_{i,\psi} \int e_\psi dt - K_{d,\psi} r \tag{42}$$

Rate limiting:

Rate commands must be within achievable limits:

$$p_{cmd} \in [-p_{\max}, +p_{\max}], \quad \text{typically } p_{\max} = 180/\text{s} \tag{43}$$

$$q_{cmd} \in [-q_{\max}, +q_{\max}], \quad \text{typically } q_{\max} = 180/\text{s} \tag{44}$$

$$r_{cmd} \in [-r_{\max}, +r_{\max}], \quad \text{typically } r_{\max} = 90/\text{s} \tag{45}$$

4.5 Gain Tuning for Attitude Controllers

The **inner loop (Level 4) must be tuned first**, then the outer loop (Level 3). This ensures the inner loop is fast and tight before the outer loop commands it.

Frequency separation rule:

$$\omega_{\text{outer}} \leq \frac{1}{10} \omega_{\text{inner}} \tag{46}$$

If the inner loop has 10 Hz bandwidth, the outer loop should have ≤ 1 Hz bandwidth.

Outer loop tuning:

- Start with $K_i = 0$, $K_d = 0$
- Increase K_p until response is smooth with $\sim$10% overshoot
- Add K_d to reduce overshoot (typically $K_d \approx 0.05K_p$ for outer loop)
- Add K_i sparingly to eliminate offset (typically $K_i \approx 0.01K_p$)

Typical values for RC aircraft:

- Roll angle: $K_p \approx 5.0$, $K_i \approx 0.5$, $K_d \approx 0.3$
- Pitch angle: $K_p \approx 6.0$, $K_i \approx 0.6$, $K_d \approx 0.4$
- Yaw angle: $K_p \approx 3.0$, $K_i \approx 0.3$, $K_d \approx 0.2$

5 Level 2: HSA Control

5.1 Flight State Variables

Level 2 controls three flight state variables:

- **Heading (H)**: ψ (rad), compass direction (0 = North, $\pi/2$ = East)
- **Speed (S)**: V (m/s), airspeed magnitude
- **Altitude (A)**: h (m), height above reference

These are the primary variables for autonomous flight missions.

5.2 HSA to Attitude Mapping

Level 2 converts HSA commands to attitude commands for Level 3.

5.2.1 Heading Control via Coordinated Turn

To change heading, the aircraft must bank (roll) and turn. For a coordinated turn:

$$\phi_{cmd} = \arctan\left(\frac{V^2}{gR}\right) \approx \frac{V\dot{\psi}}{g} \tag{47}$$

where R is turn radius and $g = 9.81$ m/s^2.
A PID controller on heading error produces yaw rate, which maps to roll angle:

$$\psi_{\text{error}} = \psi_{cmd} - \psi \quad \text{(wrapped to } [-\pi, \pi]) \tag{48}$$

$$r_{\text{des}} = K_{p,H}\psi_{\text{error}} \tag{49}$$

$$\phi_{cmd} = K_{\phi r} r_{\text{des}} \quad \text{(typically } K_{\phi r} \approx 0.5) \tag{50}$$

5.2.2 Altitude Control via Pitch

Altitude error drives climb rate, which maps to pitch angle:

$$h_{\text{error}} = h_{cmd} - h \tag{51}$$

$$\dot{h}_{\text{des}} = K_{p,A} h_{\text{error}} \quad (\text{clamped to } \pm 5 \text{ m/s}) \tag{52}$$

$$\theta_{cmd} \approx K_{h\theta} \dot{h}_{\text{des}} \quad (\text{typically } K_{h\theta} \approx 0.1) \tag{53}$$

5.2.3 Speed Control via Throttle

Speed error maps directly to throttle:

$$V_{\text{error}} = V_{cmd} - V \tag{54}$$

$$\delta_t = \delta_{t,\text{baseline}} + K_{p,S} V_{\text{error}} + K_{i,S} \int V_{\text{error}} dt \tag{55}$$

where $\delta_{t,\text{baseline}} \approx 0.5$ is cruise throttle.

5.3 HSA Controller Architecture

The complete Level 2 controller:

Algorithm 1 HSA Control Loop (10–50 Hz)

Input: $\psi_{cmd}, V_{cmd}, h_{cmd}$, current state
Output: $\phi_{cmd}, \theta_{cmd}, \psi_{cmd}, \delta_t$

$\psi_{\text{error}} \leftarrow \text{wrap}(\psi_{cmd} - \psi)$
$\phi_{cmd} \leftarrow \text{HeadingPID}(\psi_{\text{error}})$ ▷ Maps to roll

$h_{\text{error}} \leftarrow h_{cmd} - h$
$\theta_{cmd} \leftarrow \text{AltitudePID}(h_{\text{error}})$ ▷ Maps to pitch

$V_{\text{error}} \leftarrow V_{cmd} - V$
$\delta_t \leftarrow \text{SpeedPID}(V_{\text{error}})$ ▷ Throttle

Send $(\phi_{cmd}, \theta_{cmd}, \psi_{cmd}, \delta_t)$ to Level 3

6 Level 1: Waypoint Navigation

6.1 Guidance Algorithms

Level 1 converts 3D waypoint positions to HSA commands using guidance algorithms.

6.1.1 Line-of-Sight (LOS) Guidance

The simplest guidance law: point directly at the waypoint.

Given current position $\boldsymbol{p}_{\text{curr}} = [N, E, D]^T$ and waypoint $\boldsymbol{p}_{\text{wp}} = [N_{\text{wp}}, E_{\text{wp}}, D_{\text{wp}}]^T$:

$$\Delta N = N_{\text{wp}} - N \tag{56}$$
$$\Delta E = E_{\text{wp}} - E \tag{57}$$
$$\psi_{cmd} = \text{atan2}(\Delta E, \Delta N) \tag{58}$$
$$h_{cmd} = -D_{\text{wp}} \quad \text{(altitude positive up)} \tag{59}$$
$$V_{cmd} = V_{\text{cruise}} \quad \text{(or waypoint-specified speed)} \tag{60}$$

Waypoint acceptance: Declare waypoint reached when:

$$\sqrt{(\Delta N)^2 + (\Delta E)^2 + (\Delta D)^2} < R_{\text{accept}} \tag{61}$$

where typically $R_{\text{accept}} = 10$ m.

6.1.2 Proportional Navigation (Advanced)

For more aggressive pursuit, proportional navigation commands acceleration perpendicular to line-of-sight:

$$\boldsymbol{a}_{\perp} = N' V_{\text{closing}} \dot{\boldsymbol{\lambda}} \tag{62}$$

where $N' \in [3, 5]$ is navigation constant, V_{closing} is closing velocity, and $\dot{\boldsymbol{\lambda}}$ is line-of-sight rate.

This is more complex but provides tighter tracking of moving targets.

7 Level 5: Direct Surface Control

Level 5 is direct surface actuation, primarily for:

- Reinforcement learning research (learn from scratch)
- Novel aircraft configurations (no established control laws)
- Control allocation optimization

At this level, the agent directly commands $\delta_a, \delta_e, \delta_r, \delta_t \in [-1, 1]$, which are mapped to actuator commands (e.g., servo PWM signals).

This is the lowest abstraction, offering maximum control authority but requiring the agent to understand full aircraft dynamics.

8 Mathematical Foundations

8.1 Frequency Domain Analysis

Cascaded control stability relies on frequency separation. In the Laplace domain, the closed-loop transfer function of the cascaded system is:

$$G_{\text{cascade}}(s) = \frac{G_{\text{outer}}(s)G_{\text{inner}}(s)}{1 + G_{\text{inner}}(s)H_{\text{inner}}(s)} \cdot \frac{1}{1 + G_{\text{outer}}(s)H_{\text{outer}}(s)} \tag{63}$$

If the inner loop bandwidth $\omega_{\text{inner}} \gg \omega_{\text{outer}}$, the loops decouple and can be designed independently.

Bode plot criterion: The inner loop crossover frequency should be $10\times$ higher than the outer loop for clean separation.

8.2 Stability via Small Gain Theorem

For cascaded loops, the small gain theorem ensures stability if each loop is individually stable and coupling is weak:

$$\|G_{\text{coupling}}\|_\infty < \frac{1}{\|G_{\text{loop}}\|_\infty} \tag{64}$$

This justifies the decoupled multi-axis design.

9 Implementation Details

9.1 Software Architecture

The control hierarchy is implemented as a hybrid Python/C++ system:

- **Levels 1–3**: Python (flexibility for research, RL integration)
- **Levels 4–5**: C++ (performance for high-frequency control)
- **Interface**: Pybind11 for zero-copy data transfer

File structure:

```
controllers/
  base_agent.py           # Abstract agent interface
  rate_agent.py           # Level 4 (Python wrapper)
  attitude_agent.py       # Level 3 (cascaded outer loop)
  hsa_agent.py            # Level 2
  waypoint_agent.py       # Level 1

cpp/
  src/pid_controller.cpp  # C++ PID implementation
  bindings/bindings.cpp   # Pybind11 interface
```

9.2 Real-Time Execution

Control loops run at different priorities:

Table 3: Thread Priorities and Timing

Thread	Frequency	Priority	Latency Budget
Level 4 (C++)	1000 Hz	Highest	¡ 1 ms
Level 3 (C++)	100 Hz	High	¡ 10 ms
Level 2 (Python)	50 Hz	Medium	¡ 20 ms
Level 1 (Python)	10 Hz	Low	¡ 100 ms
Logging	Async	Lowest	Best-effort

10 Comparison to Industry Standards

10.1 dRehmFlight Architecture

dRehmFlight [1] implements a two-loop cascaded structure:

- **Angle mode**: Outer PID on Euler angles → rate commands
- **Rate mode**: Inner PID on angular rates → surface deflections

This is exactly our Levels 3–4. We extend dRehmFlight by adding:

- Level 2 (HSA) for autonomous flight
- Level 1 (Waypoint) for mission planning
- Level 5 (explicit surface layer) for RL research

10.2 Comparison Table

Table 4: Control Architecture Comparison

Feature	dRehmFlight	Betaflight	This Work
Angle mode	✓	✓	✓(L3)
Rate mode	✓	✓	✓(L4)
HSA control	–	Limited	✓(L2)
Waypoint nav	–	–	✓(L1)
Language	C++	C	Python + C++
Loop rate (inner)	500 Hz	8000 Hz	1000 Hz
Loop rate (outer)	100 Hz	500 Hz	100 Hz
License	GPL v3	GPL v3	(Project-specific)

Acknowledgment: Our core cascaded design (Levels 3–4) is directly inspired by and follows the proven architecture of dRehmFlight.

11 Conclusion

This document has presented a comprehensive five-level hierarchical control architecture for fixed-wing aircraft, combining the proven cascaded angle-rate structure of dRehmFlight with additional high-level layers for autonomous flight and reinforcement learning research.

Key contributions:

1. Rigorous mathematical foundations for each control level
2. Extension of dRehmFlight's two-loop design to five levels
3. Detailed implementation guidance (C++ and Python)
4. Comparison to industry-standard flight controllers
5. Suitability for both simulation and hardware deployment

Acknowledgments: This architecture owes a significant intellectual debt to Nicholas Rehm's dRehmFlight project, which established the clean separation between angle and rate control that forms the foundation of modern flight controllers. We gratefully acknowledge this inspiration while noting that our implementation is original work.

The five-level architecture provides flexibility for researchers and practitioners to choose the appropriate abstraction level for their application, from high-level mission planning down to low-level surface optimization.

References

[1] Rehm, Nicholas (2020). *dRehmFlight: Teensy Flight Controller for VTOL Aircraft.* GitHub repository. `https://github.com/nickrehm/dRehmFlight`. Licensed under GNU GPL v3.0.

[2] Betaflight Contributors (2023). *Betaflight: Open Source Flight Controller Firmware.* `https://github.com/betaflight/betaflight`.

[3] ArduPilot Development Team (2023). *ArduPilot: Open Source Autopilot Software.* `https://ardupilot.org`.

[4] PX4 Development Team (2023). *PX4 Autopilot: Professional Autopilot.* `https://px4.io`.

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

[6] Etkin, B., & Reid, L. D. (2005). *Dynamics of Flight: Stability and Control* (3rd ed.). John Wiley & Sons.

[7] Nelson, R. C. (1998). *Flight Stability and Automatic Control* (2nd ed.). McGraw-Hill.

[8] Åström, K. J., & Hägglund, T. (2006). *Advanced PID Control*. ISA-The Instrumentation, Systems, and Automation Society.

[9] Franklin, G. F., Powell, J. D., & Emami-Naeini, A. (2019). *Feedback Control of Dynamic Systems* (8th ed.). Pearson.