# StampFly Hovering 制御系・PID 詳細ドキュメント

このドキュメントは、StampFly_Hovering_Project における制御系（姿勢・高度・推力）の構成と、PID クラスがどのように利用されているかを、数式とコード位置の対応付きで整理したものです。

---

## 1. 制御系の全体構成

- メインループ周期
  - 制御ループは 400 Hz (Control_period ≒ 0.0025 [s]) で動作します。
  - 実装箇所: src/flight_control.cpp:266 付近 loop_400Hz()
- 主な状態量（センサ・推定値）
  - 姿勢角: Roll_angle, Pitch_angle, Yaw_angle [rad]
  - 角速度: Roll_rate, Pitch_rate, Yaw_rate [rad/s]
  - 高度: Altitude2 [m]（ToF + 加速度のカルマンフィルタ推定）
  - 垂直速度: Alt_velocity [m/s]
  - 実装箇所: センサ処理 src/sensor.cpp:220–335 付近
- 主な制御ループ（カスケード構造）
  1. **角度ループ（外側）**
     - 入力: 目標ロール角 / ピッチ角 Roll_angle_command, Pitch_angle_command
     - 出力: 目標角速度 Roll_rate_reference (p_ref), Pitch_rate_reference (q_ref)
     - PID: phi_pid（ロール角）、theta_pid（ピッチ角）
     - 実装: angle_control() (src/flight_control.cpp:751–782)
  2. **角速度ループ（内側）**
     - 入力: 上記の目標角速度 Roll_rate_reference, Pitch_rate_reference と Yaw_rate_reference
     - 出力: 各軸の角速度指令 Roll_rate_command, Pitch_rate_command, Yaw_rate_command
     - PID: p_pid（ロールレート）、q_pid（ピッチレート）、r_pid（ヨーレート）
     - 実装: rate_control() (src/flight_control.cpp:614–703)
  3. **高度／垂直速度ループ（外側 → 内側の二重ループ）**
     - 高度ループ（外側）
       - 入力: 高度目標 Alt_ref と推定高度 Altitude2
       - 出力: 垂直速度目標 Z_dot_ref
       - PID: alt_pid
       - 実装: angle_control() 内 (src/flight_control.cpp:760–762)
     - 垂直速度ループ（内側）
       - 入力: Z_dot_ref と推定垂直速度 Alt_velocity

    - 出力: 推力補正量を含む推力指令、最終的に `Thrust_command`
    - PID: `z_dot_pid`
    - 実装: `rate_control()` 内 (`src/flight_control.cpp:643–653`)

4. **モータミキシング**

    - 入力: `Thrust_command` と各軸角速度指令 `Roll_rate_command, Pitch_rate_command, Yaw_rate_command`
    - 出力: 各モータ duty（0–1 の正規化）
        - `FrontRight_motor_duty, FrontLeft_motor_duty, RearRight_motor_duty, RearLeft_motor_duty`
    - 実装: `rate_control()` 内 (`src/flight_control.cpp:658–671`)

---

## 2. PID クラスの実装（共通仕様）

- クラス定義: `src/pid.hpp:15` 付近 `class PID`
- 実装: `src/pid.cpp:40–80`

### 2.1 パラメータと内部状態

`PID` クラスは以下のパラメータを持ちます（`PID::set_parameter()` により設定）。

- 比例ゲイン: `m_kp = K_p`
- 積分時間: `m_ti = T_i`
- 微分時間: `m_td = T_d`
- 微分フィルタ係数: `m_eta = η`（不完全微分のフィルタ係数）
- サンプリング周期: `m_h`（`update()` 呼び出し時の `h` が優先）

内部状態:

- `m_integral` … 積分状態（無次元、誤差をスケーリングした内部積分）
- `m_differential` … 微分状態（無次元、誤差をスケーリングした内部微分）
- `m_err` … 直前ステップの誤差 `e_{k-1}`

### 2.2 離散時間 PID 方程式

更新関数: `float PID::update(float err, float h)`
実装: `src/pid.cpp:67–79`

離散時間ステップ k において、

- `e_k`: 現在の誤差 `err`
- `e_{k-1}`: 前回の誤差 `m_err`
- `h_k`: サンプル時間 `h`

とおくと、内部状態は次のように更新されます。

1. **積分項（台形則 + 1/T_i スケーリング）**

```
I_k = I_{k-1} + (h_k / (2 T_i)) (e_k + e_{k-1})
```

コードでは:

```
m_integral = m_integral + m_h * (err + m_err) / 2 / m_ti;
// ±30000 にクリップ
```

よって PID 出力に寄与する積分項は

```
u_I(k) = K_p · I_k
```

2. **不完全微分項（1 次フィルタ付き微分）**

```
D_k = α · D_{k-1} + β · (e_k − e_{k-1})

α = (2 η T_d − h_k) / (2 η T_d + h_k)
β = 2 T_d / (2 η T_d + h_k)
```

コード上は:

```
m_differential = (2 * m_eta * m_td - m_h) * m_differential
                 / (2 * m_eta * m_td + m_h)
               + 2 * m_td * (err - m_err)
                 / (2 * m_eta * m_td + m_h);
```

よって PID 出力に寄与する微分項は

```
u_D(k) = K_p · D_k
```

3. **比例項 + 積分項 + 微分項の合成**

最終出力 u_k は

```
u_P(k) = K_p · e_k
u_I(k) = K_p · I_k
u_D(k) = K_p · D_k

u_k = u_P(k) + u_I(k) + u_D(k)
    = K_p (e_k + I_k + D_k)
```

コード:

```
return m_kp * (err + m_integral + m_differential);
```

---

## 3. 姿勢角 PID（外側ループ）

関数: `angle_control()`
実装位置: `src/flight_control.cpp:751–782`

### 3.1 ロール角 PID（phi_pid）

- PID インスタンス: `phi_pid` (`src/flight_control.cpp:194`)
- ゲイン設定: `control_init()` 内 (`src/flight_control.cpp:476–477`)

```
phi_pid.set_parameter(Rall_angle_kp, Rall_angle_ti,
                      Rall_angle_td, Rall_angle_eta, Control_period);
```

ここで
$K_p$ = `Rall_angle_kp`、$T_i$ = `Rall_angle_ti`、$T_d$ = `Rall_angle_td`、$\eta$ = `Rall_angle_eta`。

**入力誤差**

`angle_control()` 内の誤差定義（`src/flight_control.cpp:765–777`）:

```
Roll_angle_reference  = Roll_angle_command;   // 目標ロール角
...
phi_err = Roll_angle_reference
          - (Roll_angle - Roll_angle_offset);
```

したがって離散時間ステップ k でのロール角誤差は

```
φ_ref(k) = Roll_angle_reference(k)
φ(k)     = Roll_angle(k) - Roll_angle_offset

e_φ(k) = φ_ref(k) - φ(k)
```

**出力**

`phi_pid` の出力はロール角速度の目標値 `Roll_rate_reference`:

```
Roll_rate_reference = phi_pid.update(phi_err, Interval_time);
```

数式で書くと、PID 出力を `p_ref(k)` として

```
p_ref(k) = Roll_rate_reference(k)
         = K_p,φ (e_φ(k) + I_φ(k) + D_φ(k))
```

ここで `I_φ(k), D_φ(k)` はセクション 2 の PID 方程式に従うロール角用の内部積分・微分状態です。

### 3.2 ピッチ角 PID（theta_pid）

- PID インスタンス: `theta_pid` (`src/flight_control.cpp:195`)
- ゲイン設定: `control_init()` 内 (`src/flight_control.cpp:478–479`)

```
theta_pid.set_parameter(Pitch_angle_kp, Pitch_angle_ti,
                        Pitch_angle_td, Pitch_angle_eta, Control_period);
```

**入力誤差**

```
Pitch_angle_reference = Pitch_angle_command;  // 目標ピッチ角
...
theta_err = Pitch_angle_reference
            - (Pitch_angle - Pitch_angle_offset);
```

離散時間ステップ k で

```
θ_ref(k) = Pitch_angle_reference(k)
θ(k)     = Pitch_angle(k) - Pitch_angle_offset

e_θ(k) = θ_ref(k) - θ(k)
```

**出力**

```
Pitch_rate_reference = theta_pid.update(theta_err, Interval_time);
```

```
q_ref(k) = Pitch_rate_reference(k)
         = K_p,θ (e_θ(k) + I_θ(k) + D_θ(k))
```

ここで `q_ref(k)` が内側のピッチ角速度ループの目標値になります。

---

# 4. 角速度 PID（内側ループ）

関数: `rate_control()`
実装位置: `src/flight_control.cpp:614–703`

## 4.1 ロールレート PID（p_pid）

- PID インスタンス: `p_pid` (`src/flight_control.cpp:191`)
- ゲイン定義: `Roll_rate_kp, Roll_rate_ti, Roll_rate_td, Roll_rate_eta`
  - 位置: `src/flight_control.cpp:84–87`
- ゲイン設定: `control_init()` (`src/flight_control.cpp:469–470`)

```
p_pid.set_parameter(Roll_rate_kp, Roll_rate_ti,
                    Roll_rate_td, Roll_rate_eta, Control_period);
```

**入力誤差**

`rate_control()` 内（`src/flight_control.cpp:635–641`）:

```
p_rate = Roll_rate;                // 実測ロール角速度
p_ref  = Roll_rate_reference;      // 目標ロール角速度
...
p_err = p_ref - p_rate;
```

よって

```
e_p(k) = p_ref(k) - p(k)
       = Roll_rate_reference(k) - Roll_rate(k)
```

**出力**

```
Roll_rate_command = p_pid.update(p_err, Interval_time);
```

```
u_p(k) = Roll_rate_command(k)
       = K_p,p (e_p(k) + I_p(k) + D_p(k))
```

この `u_p(k)` が、モータミキシング段でロール方向のトルク指令として利用されます。

## 4.2 ピッチレート PID（q_pid）

- PID インスタンス: `q_pid` (`src/flight_control.cpp:192`)

- ゲイン定義: `Pitch_rate_kp, Pitch_rate_ti, Pitch_rate_td, Pitch_rate_eta` (`src/flight_control.cpp:89–92`)
- ゲイン設定: `control_init()` (`src/flight_control.cpp:471–472`)

```
q_pid.set_parameter(Pitch_rate_kp, Pitch_rate_ti,
                    Pitch_rate_td, Pitch_rate_eta, Control_period);
```

**入力誤差**

```
q_rate = Pitch_rate;
q_ref  = Pitch_rate_reference;
...
q_err = q_ref - q_rate;
```

```
e_q(k) = q_ref(k) - q(k)
       = Pitch_rate_reference(k) - Pitch_rate(k)
```

**出力**

```
Pitch_rate_command = q_pid.update(q_err, Interval_time);
```

```
u_q(k) = Pitch_rate_command(k)
       = K_p,q (e_q(k) + I_q(k) + D_q(k))
```

### 4.3 ヨーレート PID（r_pid）

- PID インスタンス: `r_pid` (`src/flight_control.cpp:193`)
- ゲイン定義: `Yaw_rate_kp, Yaw_rate_ti, Yaw_rate_td, Yaw_rate_eta` (`src/flight_control.cpp:94–97`)
- ゲイン設定: `control_init()` (`src/flight_control.cpp:473`)

```
r_pid.set_parameter(Yaw_rate_kp, Yaw_rate_ti,
                    Yaw_rate_td, Yaw_rate_eta, Control_period);
```

**入力誤差**

`Yaw_angle_command` は常に 0 に設定され、ヨー角そのものは PID で制御されていません。
ヨーレート指令 `Yaw_rate_reference` は 0 に設定されています（`get_command()` 内,

src/flight_control.cpp:571–572 付近）。

```
r_rate = Yaw_rate;
r_ref  = Yaw_rate_reference;  // 通常 0
...
r_err = r_ref - r_rate;
```

```
e_r(k) = r_ref(k) - r(k)
       = Yaw_rate_reference(k) - Yaw_rate(k)
     ≒ - Yaw_rate(k)
```

**出力**

```
Yaw_rate_command = r_pid.update(r_err, Interval_time);
```

```
u_r(k) = Yaw_rate_command(k)
       = K_p,r (e_r(k) + I_r(k) + D_r(k))
```

---

## 5. 高度・垂直速度 PID（カスケード）

### 5.1 高度 PID（alt_pid – 外側ループ）

- PID インスタンス: alt_pid (src/flight_control.cpp:198)
- ゲイン定義: alt_kp, alt_ti, alt_td, alt_eta (src/flight_control.cpp:111–114)
- ゲイン設定: control_init() (src/flight_control.cpp:482)

```
alt_pid.set_parameter(alt_kp, alt_ti, alt_td,
                      alt_eta, alt_period);
```

**入力誤差**

angle_control() 内 (src/flight_control.cpp:760–762) にて

```
alt_err = Alt_ref - Altitude2;
if (Alt_flag >= 1)
    Z_dot_ref = alt_pid.update(alt_err, Interval_time);
```

よって

```
z_ref(k) = Alt_ref(k)          // 目標高度 [m]
z(k)     = Altitude2(k)        // 推定高度 [m]

e_alt(k) = z_ref(k) - z(k)
         = Alt_ref(k) - Altitude2(k)
```

出力は垂直速度目標 `Z_dot_ref`:

```
Ż_ref(k) = Z_dot_ref(k)
          = K_p,alt (e_alt(k) + I_alt(k) + D_alt(k))
```

### 5.2 垂直速度 PID（z_dot_pid – 内側ループ）

- PID インスタンス: `z_dot_pid` (`src/flight_control.cpp:199`)
- ゲイン定義: `z_dot_kp, z_dot_ti, z_dot_td, z_dot_eta` (`src/flight_control.cpp:117–120`)
- ゲイン設定: `control_init()` (`src/flight_control.cpp:483`)

```
z_dot_pid.set_parameter(z_dot_kp, z_dot_ti,
                        z_dot_td, alt_eta, alt_period);
```

**入力誤差**

通常ホバリング・上昇／下降時 (`Alt_flag == 1`) の場合、`rate_control()` 内で

```
z_dot_err      = Z_dot_ref - Alt_velocity;
Thrust_command = Thrust_filtered.update(
    (Thrust0 + z_dot_pid.update(z_dot_err, Interval_time))
    * BATTERY_VOLTAGE,
    Interval_time);
```

よって

```
v_ref(k) = Z_dot_ref(k)       // 外側高度 PID からの速度目標 [m/s]
v(k)     = Alt_velocity(k)    // 推定垂直速度 [m/s]

e_z(k) = v_ref(k) - v(k)
       = Z_dot_ref(k) - Alt_velocity(k)
```

自動着陸時 (`auto_state == AUTO_LANDING`) には目標速度が固定で

```
z_dot_err = -0.15 - Alt_velocity;
```

すなわち

```
v_ref(k) = -0.15 [m/s]
e_z(k)   = -0.15 - Alt_velocity(k)
```

**出力と推力指令**

`z_dot_pid` の出力を `ΔThrust(k)` とおくと

```
ΔThrust(k) = z_dot_pid(e_z(k))
```

これに基準推力 `Thrust0` を加算し、電圧スケーリングを行った上で一次遅れフィルタ `Thrust_filtered` を通して最終推力指令 `Thrust_command` を計算しています。

```
Thrust_raw(k) = [Thrust0(k) + ΔThrust(k)] · BATTERY_VOLTAGE
Thrust_command(k) = LPF_Thrust(Thrust_raw(k))
```

コード:

```
Thrust_command = Thrust_filtered.update(
    (Thrust0 + z_dot_pid.update(z_dot_err, Interval_time))
    * BATTERY_VOLTAGE,
    Interval_time);
```

`Thrust0` はバッテリ電圧から計算されるトリム推力で、`get_command()` 内で更新されています (`src/flight_control.cpp:506–521`)。

さらに、推力の急激な変化を抑えるために上下 15% の範囲でクリップしています (`src/flight_control.cpp:648–649`)。

```
Thrust_command / BATTERY_VOLTAGE ∈ [0.85·Thrust0, 1.15·Thrust0]
```

---

## 6. モータミキシングと最終出力

`rate_control()` 内のモータ duty 計算部（`src/flight_control.cpp:658–671`）では、

```
FrontRight_motor_duty = Duty_fr.update(
    (Thrust_command + (-Roll_rate_command
                       + Pitch_rate_command
                       + Yaw_rate_command) * 0.25f)
    / BATTERY_VOLTAGE,
    Interval_time);

FrontLeft_motor_duty = Duty_fl.update(
    (Thrust_command + (Roll_rate_command
                       + Pitch_rate_command
                       - Yaw_rate_command) * 0.25f)
    / BATTERY_VOLTAGE,
    Interval_time);

RearRight_motor_duty = Duty_rr.update(
    (Thrust_command + (-Roll_rate_command
                       - Pitch_rate_command
                       - Yaw_rate_command) * 0.25f)
    / BATTERY_VOLTAGE,
    Interval_time);

RearLeft_motor_duty = Duty_rl.update(
    (Thrust_command + (Roll_rate_command
                       - Pitch_rate_command
                       + Yaw_rate_command) * 0.25f)
    / BATTERY_VOLTAGE,
    Interval_time);
```

を計算しています。

各 `Duty_*` は `Filter` クラス（一次遅れフィルタ）で平滑化されており、その実装は `src/pid.cpp:82–101` にあります。

### 6.1 モータ出力の数式化

簡略化のため、フィルタを無視した瞬時値として、

```
u_T(k) = Thrust_command(k) / BATTERY_VOLTAGE
u_p(k) = Roll_rate_command(k)
u_q(k) = Pitch_rate_command(k)
u_r(k) = Yaw_rate_command(k)
```

とすると、各モータ duty（0–1 の正規化値）は概ね以下の線形結合です。

```
FrontRight ≒ u_T + (-u_p + u_q + u_r) / 4
FrontLeft  ≒ u_T + ( u_p + u_q - u_r) / 4
RearRight  ≒ u_T + (-u_p - u_q - u_r) / 4
RearLeft   ≒ u_T + ( u_p - u_q + u_r) / 4
```

この duty が `set_duty_fr/fl/rr/rl()` を通して PWM に変換され、最終的にモータに出力されます (`src/flight_control.cpp:785–796`)。

---

## 7. 実装箇所まとめ（ファイル・関数・行番号）

- PID クラス本体
  - ファイル: `src/pid.cpp:40–80`
  - 関数: `PID::set_parameter()`, `PID::update()`
  - 内容: 共通の P, I, D 計算ロジック
- PID パラメータ定義
  - ファイル: `src/flight_control.cpp:82–120`
  - 内容:
    - ロール・ピッチ・ヨー角速度 PID ゲイン (`Roll_rate_*`, `Pitch_rate_*`, `Yaw_rate_*`)
    - ロール・ピッチ角度 PID ゲイン (`Rall_angle_*`, `Pitch_angle_*`)
    - 高度 PID (`alt_*`) と垂直速度 PID (`z_dot_*`) ゲイン
- PID インスタンスと初期化
  - ファイル: `src/flight_control.cpp:191–203` (`PID p_pid, q_pid, r_pid, phi_pid, theta_pid, alt_pid, z_dot_pid;`)
  - ゲイン設定: `control_init()` (`src/flight_control.cpp:467–483`)
- 角度 PID（外側ループ）
  - ファイル: `src/flight_control.cpp:751–782`
  - 関数: `angle_control()`
  - 内容:
    - 高度誤差 `alt_err = Alt_ref - Altitude2` → `Z_dot_ref = alt_pid.update(...)`
    - 角度誤差 `phi_err, theta_err` → `Roll_rate_reference, Pitch_rate_reference = phi_pid/theta_pid.update(...)`
- 角速度 PID（内側ループ）
  - ファイル: `src/flight_control.cpp:614–703`
  - 関数: `rate_control()`
  - 内容:
    - 角速度誤差 `p_err, q_err, r_err` → `Roll_rate_command, Pitch_rate_command, Yaw_rate_command`
    - 垂直速度誤差 `z_dot_err` → `Thrust_command`（`z_dot_pid` と一次遅れ `Thrust_filtered` により）
- 高度推定・センサ処理
  - ファイル: `src/sensor.cpp:220–335`
  - 関数: `sensor_read()`

- 内容:
  - ToF + 加速度から Kalman フィルタ `EstimatedAltitude` (`Alt_kalman`) を用いて `Altitude2`, `Alt_velocity`, `Az_bias` を算出
  - これらが高度／垂直速度 PID の入力となる

---

このドキュメントを起点に、各 PID のゲイン調整や制御設計（例: 角度ループの帯域、レートループの帯域、高度ループの応答性など）を検討する際には、セクション 2 の離散時間 PID 方程式と、セクション 3–5 の誤差定義・信号の流れを参照すると、コードと制御理論の対応付けが容易になります。