Attitude Control for Quadcopters: A PID-Based Approach

Akash Ajin, Akhilesh Menon, Paul Choi, Shiv Khannade
MAT292: Ordinary Differential Equations

Fall 2025

Abstract

This project investigated the attitude dynamics of a quadcopter, an unstable system that requires active control for stable flight. We developed a mathematical model based on Newton-Euler equations, designed a Proportional-Integral-Derivative (PID) control system to stabilize it, and evaluated its performance through numerical simulation in Python. This work demonstrates the practical application of solving systems of ordinary differential equations (ODEs) to overcome real-world engineering challenges, highlighting their importance.

1 Introduction

1.1 Motivation

A quadcopter is an unmanned aerial vehicle (UAV) whose flight is controlled by four motors. This UAV's navigational agility has made it popular in fields from photography to logistics. However, quadcopters are inherently unstable. Without a constant stream of adjustments from a control system, minor disturbances would cause them to tumble. The system's dynamics are also highly non-linear and coupled, making it a very fitting problem for an ODE course.

1.2 Project Goal

The primary objective of this project was to model the unstable flight dynamics of a quadcopter as a system of coupled, non-linear ordinary differential equations. We then designed and simulated a PID feedback controller to impose stability, allowing the UAV to achieve a stable hover and reject external disturbances. The project provides a clear, practical demonstration of how ODEs and control theory are used to solve a fundamental problem in modern robotics.

2 Mathematical Model and Theoretical Foundation

The quadcopter's motion is modeled as a rigid body in 3D space, using a fixed inertial frame (Earth, E) and a rotating body frame (Body, B) attached to the vehicle.

2.1 State Vector

The system is described by a 12-element state vector $\mathbf{x} \in \mathbb{R}^{12}$:

$$\mathbf{x} = [\ \underbrace{x, y, z}_{\substack{\text{position} \\ \text{(Frame E)}}}\ ,\quad \underbrace{\phi, \theta, \psi}_{\substack{\text{attitude (Euler)} \\ \text{(Frame E)}}}\ ,\quad \underbrace{\dot{x}, \dot{y}, \dot{z}}_{\substack{\text{lin. velocity} \\ \text{(Frame E)}}}\ ,\quad \underbrace{p, q, r}_{\substack{\text{ang. velocity} \\ \text{(Frame B)}}}\ ]^T$$

This state is a hybrid, as linear motion is tracked in the inertial frame E while rotational motion is tracked in the body frame B. This requires transformation matrices to couple the dynamics.

2.2 Translational Dynamics

In the inertial frame, Newton's 2nd Law governs translational motion:

$$\mathbf{F}_{\text{net},E} = m\ddot{\mathbf{p}}_E = m \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix}$$

The net force is the sum of gravity $\mathbf{F}_{grav,E} = [0, 0, -mg]^T$ and the total thrust $\mathbf{F}_{thrust,B} = [0, 0, T]^T$. Thrust is generated in the body frame (acting along the quadcopter's z_B-axis) and must be rotated into the inertial frame:

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \frac{1}{m} \left(\begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R(\phi, \theta, \psi) \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} \right)$$

where $T = T_1 + T_2 + T_3 + T_4$ and R is the $Z - Y - X$ body-to-inertial rotation matrix. R is constructed as $R = R_z(\psi)R_y(\theta)R_x(\phi)$:

$$R = \begin{bmatrix} c\psi c\theta & c\psi s\theta s\phi - s\psi c\phi & c\psi s\theta c\phi + s\psi s\phi \\ s\psi c\theta & s\psi s\theta s\phi + c\psi c\phi & s\psi s\theta c\phi - c\psi s\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix}$$

where $c\alpha = \cos(\alpha)$ and $s\alpha = \sin(\alpha)$. The first six ODEs are thus: $\dot{x} = \dot{x}$, $\dot{y} = \dot{y}$, $\dot{z} = \dot{z}$, and the three translational acceleration equations above $(\ddot{x}, \ddot{y}, \ddot{z})$.

2.3 Rotational Dynamics

Rotational motion is described by the Newton-Euler equations in the body frame:

$$\tau_B = I\dot{\omega}_B + \omega_B \times (I\omega_B)$$

where $\tau_B = [\tau_\phi, \tau_\theta, \tau_\psi]^T$ are the net torques, I is the 3×3 inertia tensor, and $\omega_B = [p, q, r]^T$. Assuming a diagonal inertia tensor $I = \text{diag}(I_{xx}, I_{yy}, I_{zz})$, the ODEs for the angular rates are:

$$\begin{aligned}
\dot{p} &= (1/I_{xx})\left(\tau_\phi - (I_{zz} - I_{yy})qr\right) \\
\dot{q} &= (1/I_{yy})\left(\tau_\theta - (I_{xx} - I_{zz})pr\right) \\
\dot{r} &= (1/I_{zz})\left(\tau_\psi - (I_{yy} - I_{xx})pq\right)
\end{aligned}$$

The Euler angle derivatives (in frame E) are related to the body rates (in frame B) by the transformation W:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = W(\phi, \theta) \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi \tan\theta & \cos\phi \tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi / \cos\theta & \cos\phi / \cos\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

These nine equations form the complete system of 12 first-order ODEs ($\dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi}, \ddot{x}, \ddot{y}, \ddot{z}, \dot{p}, \dot{q}, \dot{r}$), which can be written in the form $\dot{\mathbf{x}} = f(t, \mathbf{x}, \mathbf{u})$.

3 Methodology

3.1 Control Strategy: PID Controller

To stabilize the unstable dynamics, we implemented four independent PID controllers. The PID control law calculates a corrective action $u(t)$ based on the error $e(t) = r(t) - y(t)$, where $r(t)$ is the desired setpoint and $y(t)$ is the measured state:

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

The Proportional (K_p) term responds to current error, the Integral (K_i) term eliminates steady-state drift, and the Derivative (K_d) term anticipates future error to dampen oscillations§. We designed one controller for altitude (z) and three for attitude (ϕ, θ, ψ).

3.2 Control Allocation

The four PID outputs, known as virtual controls, $\mathbf{u}_v = [T, \tau_\phi, \tau_\theta, \tau_\psi]^T$, must be mapped to the four individual motor thrusts $\mathbf{T}_m = [T_1, T_2, T_3, T_4]^T$. Based on the quadcopter configuration in our proposal (a '+' configuration) and assuming motor 1 is rear, 4 is front, 2 is right, and 3 is left:

$$\begin{aligned}
T &= T_1 + T_2 + T_3 + T_4 \\
\tau_\phi &= L(T_2 - T_3) \quad \text{(Roll torque)} \\
\tau_\theta &= L(T_1 - T_4) \quad \text{(Pitch torque)} \\
\tau_\psi &= k_m(T_1 - T_2 + T_3 - T_4) \quad \text{(Yaw torque)}
\end{aligned}$$

where L is the arm length and k_m is the thrust-to-torque coefficient. This can be written as $\mathbf{u}_v = M\mathbf{T}_m$. For the controller, we need the inverse mapping $\mathbf{T}_m = M^{-1}\mathbf{u}_v$:

$$\begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = \begin{bmatrix} 1/(4) & 0 & 1/(2L) & 1/(4k_m) \\ 1/(4) & 1/(2L) & 0 & -1/(4k_m) \\ 1/(4) & -1/(2L) & 0 & 1/(4k_m) \\ 1/(4) & 0 & -1/(2L) & -1/(4k_m) \end{bmatrix} \begin{bmatrix} T \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix}$$

This M^{-1} matrix is the control allocation matrix.

3.3 Numerical Simulation

The simulation was implemented in Python. We used a main loop that calls `scipy.integrate.solve_ivp` at each discrete control step ($dt = 0.01s$). At the start of each step, the PID controllers calculate the virtual controls $\mathbf{u}_v$ based on the current state $\mathbf{x}(t_i)$. These are mapped to motor thrusts $\mathbf{T}_m$ which are held constant for the duration of the step. `solve_ivp` then numerically integrates the ODE system $\dot{\mathbf{x}} = f(t, \mathbf{x}, \mathbf{T}_m)$ from t_i to t_{i+1}. This approach models a digital controller with a zero-order hold, sampling the continuous-time dynamics of the quadcopter.

4 Results

4.1 PID Tuning

The PID gains (K_p, K_i, K_d) for each of the four controllers were tuned manually using a method based on Ziegler-Nichols. First, the K_i and K_d gains were set to zero. The proportional gain K_p was then increased until the system (e.g., the roll angle) exhibited stable, sustained oscillations, noting this ultimate gain K_u and oscillation period T_u. The gains were then set using "classic PID" ratios: $K_p = 0.6K_u$, $K_i = 2K_p/T_u$, and $K_d = K_pT_u/8$. This process was repeated for the roll and pitch controllers (which share gains due to symmetry), followed by the yaw controller, and finally the altitude controller. The gains were then fine-tuned to achieve a fast, critically damped response with minimal overshoot.

4.2 Hover Stabilization

Figure 1: Altitude (z-axis) response to a 1.0m step command. The controller stabilizes the quadcopter at the desired altitude.

Figure 2: Attitude (roll/pitch) response to a simulated disturbance (e.g., a 10-degree roll command for 1 second). The plot shows the controller's ability to reject the disturbance and return to a stable hover.

4.3 Disturbance Rejection

5 Discussion

6 Conclusion

A Simulation Code

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

# ... paste your full, final Python script here ...
# (See quadcopter_simulation.py)

\end{G}
```