SMART THROTTLE CONTROL

EEA project (phase 1)

Project Progress Summary

The project has progressed from theoretical modeling to advanced controller design and hardware implementation strategies.

Key areas of focus included:

- System Modeling and Representation:
  - **Laplace Domain**: Developed the ability to represent physical systems using transfer functions G(s).
  - **Physical Systems**: Derived differential equations for mechanical systems, such as the Mass-Spring-Damper model, using Newton's Second Law.
  - **State-Space**: Learned alternative representations using internal states ($x' = Ax+Bu$) for more complex MIMO systems.

- Frequency and Time Domain Analysis:
  - **Bode Plots**: Mastered sketching and interpreting asymptotic Bode magnitude and phase plots to

understand system stability and frequency response.

- **Step Response**: Utilized MATLAB's step() and stepinfo() commands to analyze transient characteristics like Rise Time (t_r), Settling Time (t_s), and Overshoot (M_p).
- **Steady-State Accuracy**: Applied the Final Value Theorem to determine steady-state error (e_{ss}) for different system types (Type 0, 1, and 2) under step and ramp inputs.

- Controller Design and Tuning:
  - **PID Control**: Explored the roles of Proportional (P) for speed, Integral (I) for eliminating error, and Derivative (D) for damping.
  - **Compensators**: Learned to use Lead compensators to improve transient speed and Lag compensators to enhance steady-state accuracy.
  - **Advanced Techniques**: Introduced to Feedforward control for proactive response and MIMO strategies for interacting control loops

- Hardware Implementation:
  - **Digital Control**: Understood the transition from continuous analog circuits (op-amps) to digital microcontrollers using Discretization (e.g., Tustin transformation) and fixed sampling times (T_s).

Key Tools and Techniques that were used throughout the project:

- MATLAB/Octave/Python: Used for calculating poles/zeros, generating Bode plots, and simulating step responses.
- Simulink: Employed for block diagram modeling and automatic C-code generation for hardware deployment.
- Control Design Commands: Utilized tf(), bode(), step(), feedback() for system manipulation.
- Digital Interface: Knowledge of ADC (Analog-to-Digital Conversion) for sensors and PWM (Pulse Width Modulation) for actuators.

Challenges Faced:

- Parameter Correlation: Managing the interdependency of PID terms, where improving one metric (like rise time) can negatively impact another (like stability).
- System Type Constraints: Realizing that a Type 0 system cannot track a ramp input with finite error, necessitating a higher system type or specific controller adjustments.
- Real-world Nonlinearities: Bridging the gap between ideal simulations and hardware implementation factors like friction, delays, and sensor noise.

Drone Altitude and Speed Control System

I chose '**Drone altitude and speed control system**' because it is a great demonstrate of how multiple control loops work together. For a drone to remain stable while moving, it must constantly balance gravity, lift, and air resistance.

Overall Workflow & Architecture

The system operates on a **Closed-Loop Feedback** mechanism. It follows a "Sensing to Action" pipeline that repeats hundreds of times per second.

1. **Sensor Inputs**: The drone gathers data about its current state (Where am I? How fast am I moving?).
2. **Processing (The Controller)**: The "brain", which is basically the flight controller (FC) compares the current state to the user's desired state (The Setpoint) and calculates the error.
3. **Actuator Control**: The brain sends signals to the Electronic Speed Controllers (ESCs) to spin the motors faster or slower to correct that error.

Key Components & Modules

To manage both altitude and speed, the system is typically divided into specific functional blocks:

A. The Navigation & Sensing Block

- **Barometer/Ultrasonic Sensor:** Measures air pressure or distance to the ground to determine altitude.
- **GPS Module:** Tracks horizontal position to calculate **ground speed**.

B. The PID Controller (The "Core")

This is the mathematical heart of the system. It uses three parameters to reach the target altitude/speed smoothly:

- **Proportional (P):** Corrects the current error. If you only use **P**, the drone will never actually reach the target. As it gets closer, the error gets smaller, and the "push" becomes too weak to overcome gravity. This is called Steady-State Error.
- **Integral (I):** Corrects accumulated errors over time. This eliminates that "Steady-State Error" and allows the drone to hover exactly where it should, even if there is a constant force like wind pushing it down.
- **Derivative (D):** Predicts future error to prevent overshooting. his term looks at the rate of change (how fast the drone is moving toward the target). It acts like a shock absorber or a brake.

C. The Mixer Unit

Drones usually have four motors. If you want to go **up**, the mixer increases power to all four equally. If you want to go **forward (speed)**, the mixer increases power to the back motors and decreases power to the front, tilting the drone.

D. Safety & Failsafe Mechanisms

- **Battery Monitor:** Triggers an auto-land if voltage is too low.
- **Signal Loss Protection:** Commands the drone to hover or return to home if the radio link is cut.

In mathematical terms, the total output ($u(t)$) sent to the motors is the sum of these three parts:

$$u(t) = K_p * e(t) + K * \text{integral}\ \{e(t)\ dt\} + K_d * [d\{e(t)\}/dt]$$