SMART THROTTLE CONTROL SYSTEM: Theory and Robust Implementation

BHAVIT MEENA
Roll No: 240272

January 2026 — Mid-Term Evaluation

Abstract

This report synthesizes the theoretical foundations and practical implementation strategies acquired during the Smart Throttle Control project. It transitions from fundamental system modeling in the Laplace domain to advanced robust control techniques, including anti-windup architectures and digital discretization. The report culminates in a detailed design proposal for an **Adaptive Cruise Control (ACC)** system, demonstrating how feedforward-feedback mechanisms can be synthesized to handle non-linear road disturbances and actuator saturation constraints.

1 Overall Summary: Evolution of Control Understanding

Our journey through the Smart Throttle sessions was about to understanding the personality of dynamic systems. We progressed by studying ideal mathematical models to learn to create a one

1.1 1. System Modeling: The Language of Dynamics

We began by translating physical reality (masses, dampers, motors) into the mathematical domain using Transfer Functions $G(s)$.

- **The Concept:** We utilized Newton's laws to derive differential equations and applied the Laplace Transform to analyze systems in the s-domain.
- **Key Metrics:** We learned that the location of poles determines stability, while the "Type" of the system (number of integrators) dictates its ability to track inputs without steady state error in the starting phase.

Intuitive Insight: Why this matters: A transfer function isn't just algebra; it's the DNA of the system. A pole in the Right Half Plane (RHP) isn't just a math error—it represents a physical explosion or a system that breaks apart. We learned that before we can control a system, we must respect its natural physics.

1.2 2. The PID Paradigm: Past, Present, and Future

We explored the Proportional-Integral-Derivative controller, as a **temporal logic engine** that combines three distinct perspectives of time to solve control problems.

- **Proportional (K_p - The Present):** Reacts to the immediate error. High K_p improves rise time but introduces oscillation.

- **Integral (K_i - The Past):** Accumulates the history of the error. It is the only term capable of eliminating steady-state error completely by "remembering" that the target hasn't been met.
- **Derivative (K_d - The Future):** Predicts the error trend based on the rate of change. It provides "electronic damping" to reduce overshoot.

Example: The Hill Climb Scenario

To visualize these "personalities," consider our Smart Throttle vehicle attempting to maintain 60 km/h while driving up a steep hill:

1. **P-Action (The Reactive Driver):** The car slows to 55 km/h due to gravity. The Proportional term sees a 5 km/h error and applies throttle. However, as the speed nears 59 km/h, the error shrinks, and the P-term reduces throttle. It lacks the strength to push the final 1 km/h against gravity, resulting in *Steady-State Error*.
2. **I-Action (The Persistent Driver):** The Integral term notices that the error has remained non-zero for several seconds. It "gets frustrated" (mathematically integrates the error area) and ramps up the throttle output beyond what the P-term demands, forcing the vehicle up to exactly 60 km/h.
3. **D-Action (The Anticipatory Driver):** As the car crests the hill and the slope flattens, the vehicle begins to accelerate rapidly. A human driver would lift off the pedal *before* reaching the speed limit to avoid a ticket. Similarly, the Derivative term sees the positive slope of velocity ($+\frac{de}{dt}$) and applies negative correction (braking/throttle cut) *before* the overshoot occurs.

Intuitive Insight: We discovered a fundamental trade-off: The Integral term is like a stubborn perfectionist—it ensures accuracy but reacts slowly (phase lag), causing oscillation if the gain is too high. The Derivative term is like a nervous predictor—it acts fast to stop crashes (overshoot) but panics when the sensor signal is noisy. Balancing these "personalities" is the art of tuning.

1.3 3. Addressing Real-World Constraints

The most critical learning phase involved moving beyond ideal linear simulations to handle the non-linear limitations of physical hardware. In a simulation, a variable can grow to infinity; in reality, a voltage or throttle angle has a hard limit.

1.3.1 A. The Saturation Problem (Integral Windup)

The Scenario: Consider our ACC vehicle attempting to climb a steep gradient. The reference speed is 60 km/h, but due to the load, the engine physically cannot push the car past 50 km/h even at wide-open throttle (100% saturation).

- **The Failure Mechanism:** The error term $e(t) = 10$ km/h remains positive. The Integral term, defined as $I(t) = K_i \int e(t)dt$, continues to sum this error.

$$I(t) \rightarrow \text{unbounded growth (150\%, 200\%, 500\%...)}$$

  The mathematical controller is demanding "more throttle" to close the gap, unaware that the physical throttle is already hitting the mechanical stop.

- **The Consequence (Windup):** When the road finally flattens, the car accelerates. Even when $v = 60$ km/h (error = 0), the stored Integral value is still massive (e.g., 500%). The controller keeps the throttle floored until the error becomes *negative* long enough to "drain" that accumulated value. This guarantees a massive overshoot and potential safety hazard.
- **The Solution (Clamping):** We implemented a conditional integration logic:

  $$\text{If } (u_{calculated} > u_{max}) \text{ AND } (e(t) \text{ has same sign as } u), \text{ THEN Halt Integration.}$$

  This "freezes" the integrator when the actuator is saturated, preventing the build-up of phantom error.

Intuitive Insight: Think of Integral Windup like pulling back a slingshot. If you are stuck against a wall (saturation) but keep pulling the rubber band back (integrating error), you aren't moving forward. But the moment the wall disappears, that stored energy releases all at once, launching you way past your target. Clamping tells you: "Stop pulling the band if you hit the wall."

1.3.2 B. The Noise Problem (Derivative Kick)

The Scenario: Real sensors are never perfect. A wheel speed encoder might read 60.0 km/h, then 60.1 km/h, then 59.9 km/h due to vibration or quantization, even if the car is moving at a constant speed.

- **The Failure Mechanism:** The Derivative term looks at the rate of change: $\frac{de}{dt} \approx \frac{\Delta e}{\Delta t}$. If the sensor jumps 0.1 km/h in a sampling time of $0.01s$:

  $$\text{Slope} = \frac{0.1}{0.01} = 10 \text{ km/h/s}$$

  Even microscopic noise creates massive instantaneous slopes. Since $D = K_d \times \text{Slope}$, this noise is amplified into violent spikes in the control signal.
- **The Consequence:** The throttle motor receives a "jittery" signal ($0\% \rightarrow 20\% \rightarrow 0\%$), causing the actuator to overheat and the ride to be jerky, despite the average speed being constant.
- **The Solution (Low-Pass Filter):** We replaced the pure derivative s with a filtered derivative:

  $$D(s) = \frac{sK_d}{\tau s + 1}$$

  This acts as a "shock absorber" for the data, allowing the slow, true changes in speed to pass through while blocking the high-frequency sensor static.

Intuitive Insight: Imagine driving with a nervous passenger who screams "Stop!" every time they see a pebble on the road. That is a pure Derivative controller reacting to noise. The Low-Pass filter puts noise-canceling headphones on the controller, so it only reacts to the actual turns in the road (true system dynamics) and ignores the pebbles.

1.4 4. Advanced Architectures: Looking Beyond Feedback

While feedback loops are robust, we realized they are inherently *reactive*—they must wait for an error to occur before they can fix it. To achieve high-performance control, we moved to *proactive* architectures

1.4.1 A. Feedback vs. Feedforward: The Reactive vs. Proactive Debate

We established a clear distinction between the two primary control philosophies:

- **Feedback (Closed Loop):** Compares output to reference. It is safe and accurate but slow. It drives by looking at the speedometer.
- **Feedforward (Open Loop Augmentation):** Measures the disturbance directly and injects a correction signal immediately. It drives by looking at the road slope.

Intuitive Insight: Imagine walking in a dark room (Feedback). You only correct your path after you bump into a wall. Now imagine turning on the lights (Feedforward). You see the wall coming and turn *before* you hit it. Feedforward provides the "sight"; Feedback provides the "touch."

1.4.2 B. The Power of Physics: Feedforward Control

The Scenario: A car approaches a hill.

- **Feedback Approach:** The car hits the hill → Speed drops → Error increases → PID reacts → Throttle opens. *Result: Temporary speed loss.*
- **Feedforward Approach:** The IMU sensor detects the tilt angle θ *before* speed drops.

$$u_{FF} = \frac{mg\sin(\theta)}{K_{motor}}$$

The controller instantly adds enough throttle to cancel gravity. *Result: Zero speed loss.*

1.4.3 C. Compensators: Reshaping the Root Locus

Sometimes, a simple PID cannot stabilize a system because the "roots" (poles) of the system are too stubborn. We learned to use Compensators to physically move these poles [Session 4].

- **Lead Compensator (The Turbocharger):** Adds a "zero" closer to the origin than its "pole" ($|z| < |p|$). This adds *phase lead*, effectively speeding up the system response without causing instability.
- **Lag Compensator (The Stabilizer):** Adds a pole closer to the origin ($|p| < |z|$). This acts like a low-pass filter, reducing high-frequency gain to improve steady-state accuracy.

Intuitive Insight: Think of PID gains (K_p, K_d) as adjusting the volume on your stereo. Think of Compensators as the Equalizer (Bass/Treble). Sometimes turning up the volume (Gain) just causes distortion (Oscillation). You need to specifically boost the Treble (Lead) to get crispness (Speed) or boost the Bass (Lag) to get depth (Steady-State).

2 Project Implementation Plan

Selected System: Adaptive Cruise Control (ACC) with Disturbance Rejection

2.1 Design Philosophy

I have selected the Adaptive Cruise Control system to demonstrate the **Feedforward-Feedback** hybrid architecture. A standard PID loop (Feedback) is insufficient for changing terrain because it is reactive—it waits for an error (speed drop) before acting.

Our goal is to create a "Smart" throttle that:

1. **Anticipates:** Uses sensors to "feel" the hill and applies power immediately (Feedforward).
2. **Corrects:** Uses PID to fix minor speed deviations (Feedback).

2.2 1. System Architecture

The system functions as a cascaded loop where sensor data flows into a central decision core, which then commands the actuator.

2.2.1 Phase 1: Sensing Layer (Inputs)

We fuse data from three distinct physical sources:

- **Feedback Input (v_{meas}):** Hall-effect sensors on the wheels measure the actual velocity.
- **Disturbance Input (θ):** An IMU (Inertial Measurement Unit) measures the road pitch angle to detect hills.
- **Safety Input (d_{rel}):** Radar/Lidar measures the distance to the vehicle ahead.

2.2.2 Phase 2: The Control Core (Processing)

The microcontroller computes the final throttle command $u(t)$ by summing two independent paths:

$$u_{total}(t) = \underbrace{u_{FF}(t)}_{\text{Proactive}} + \underbrace{u_{FB}(t)}_{\text{Reactive}}$$

A. The Proactive Path (Feedforward): This block handles the physics of the road. Using the estimated mass (m) and road angle (θ), we calculate the gravity force component:

$$F_{gravity} = mg\sin(\theta)$$

We instantly map this force to a specific throttle percentage.

Intuitive Insight: If the sensor detects a $10°$ slope, the engine revs up *immediately.* The car maintains speed without waiting for the speedometer to drop first.

B. The Reactive Path (Feedback PID): The PID controller manages the subtle errors caused by wind resistance and tire friction.

$$u_{FB}(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{d}{dt}e(t)$$

Note: This block includes Anti-Windup logic to prevent integrator saturation during full-throttle acceleration.

2.2.3 Phase 3: Actuation Layer (Output)

The summed control signal (0-100%) is converted into a **PWM (Pulse Width Modulation)** signal. This voltage drives the DC Motor (Electronic Throttle Body) to open the air intake valve.

2.3 2. Safety Logic & Digital Implementation

2.3.1 A. The Safety State Machine

Before the control signal is generated, the system must decide *which* objective is more important: Speed or Safety. We implement a simple supervisory logic:

```
IF (Distance_to_Car_Ahead < Safe_Gap) THEN
    Target = Match_Lead_Car_Speed()   [CRITICAL SAFETY MODE]
ELSE
    Target = Maintain_Driver_Set_Speed() [NORMAL CRUISE MODE]
END IF
```

2.3.2 B. From Math to Microcontroller (Discretization)

Since a microcontroller cannot calculate a continuous integral ($\int$), we must convert the math into discrete steps. We use the **Trapezoidal (Tustin) Approximation**:

$$\text{Current Integral} = \text{Previous Integral} + \text{New Area}$$
$$I[k] = I[k-1] + \frac{K_i \cdot T_s}{2}(Error[k] + Error[k-1])$$

Intuitive Insight: Think of the integral as filling a bucket with water (error). Since the computer blinks every 10ms (T_s), we can't pour continuously. Instead, we add a small cup of water every 10ms. The formula above calculates exactly how big that cup should be.

2.4 3. Addressing Non-Linearity (Gain Scheduling)

The Problem: A car engine behaves differently at different speeds. Opening the throttle by 5% when the car is stopped produces a huge jump (high torque). Opening it by 5% when driving at 100 km/h produces very little change.

The Solution - Gain Scheduling: We do not use a single set of PID gains. We create a lookup table:

- **Low Speed:** Use **Low** K_p (to avoid jerky starts).
- **High Speed:** Use **High** K_p (to ensure the engine responds to commands).

Intuitive Insight: This is similar to how a human driver behaves. You press the pedal gently in a parking lot (Low Gain) but firmly when passing on a highway (High Gain). The controller adapts its "aggression" based on the speed.

3 Conclusion

This project moved us from the theoretical s-plane to the practical realization of control logic. We learned that a good controller is not just about mathematical stability; it is about respecting physical limits (saturation), anticipating disturbances (feedforward), and handling the noise of the real world. The proposed ACC system integrates these lessons into a robust, safety-critical architecture.