Speed: The Missing Layer in DeFi Solvency

Low-Latency Hierarchical Control for Continuous Deleveraging

-Akshaya Krishna

February 2026

Abstract

We present a hierarchical control framework for proactive solvency management in DeFi. By solving an ergodic Hamilton-Jacobi-Bellman (HJB) objective, we derive a strategic deleveraging velocity that minimizes long-term market impact and inventory risk. This velocity is tracked by a discrete-time PID controller modified with derivative filtering and integral clamping. Implementation on MagicBlock Ephemeral Rollups ensures the low-latency execution required for high-frequency risk mitigation.

The proposed framework adapts established ergodic control formulations and standard PID tracking methods to DeFi solvency management via a hierarchical, low-latency execution architecture.

1 Introduction

Current DeFi liquidation regimes suffer from a "Binary Trap," where insolvency triggers a zero-sum race between liquidators. We propose a "User Agent" model that operates in the pre-insolvency Yellow Zone ($1.0 < H < 1.05$), utilizing formal control theory to ensure smooth deleveraging.

2 Strategic Layer: Ergodic HJB Formulation

We consider a reduced-order state representation in which the dominant risk variable is the position inventory q, while price dynamics and protocol parameters are assumed locally stationary within the Yellow Zone. This approximation is valid over short control horizons and allows tractable derivation of the optimal deleveraging policy.

2.1 Objective Function

We define the state dynamics as $dq_t = -v_t\,dt$. The goal is to maximize the long-term average reward ρ:

$$\rho = \limsup_{T\to\infty} \frac{1}{T}\mathbb{E}\left[\int_0^T \left(-\eta v_t^2 - \frac{1}{2}\gamma\sigma^2 q_t^2\right) dt\right] \tag{1}$$

The corresponding HJB equation is:

$$\rho + \max_v \left\{-\eta v^2 - \frac{1}{2}\gamma\sigma^2 q^2 - v\frac{dV}{dq}\right\} = 0 \tag{2}$$

2.2 Derivation of Optimal Velocity

Applying the first-order condition $\frac{\partial}{\partial v} : -2\eta v - \frac{dV}{dq} = 0$, we find:

$$v^*(q) = -\frac{1}{2\eta}\frac{dV}{dq} \tag{3}$$

Assuming a quadratic value function $V(q) = Aq^2$ and solving for the stationary policy, the strategic target velocity simplifies to:

$$v^*(q) = \kappa q, \quad \kappa = \frac{\gamma\sigma^2}{2\eta} \tag{4}$$

3 Tactical Layer: PID Tracking Controller

Rather than directly enforcing a target solvency state, the solution of the ergodic HJB problem defines a desired deleveraging velocity, which is subsequently tracked by a lower-level controller. This hierarchical separation between strategic optimality and tactical execution is standard in high-frequency control systems.

3.1 Controller Refinements

The discrete-time PID controller is employed as a practical tracking mechanism rather than a globally optimal solver. While a formal stability proof is beyond the scope of this work, boundedness is enforced via derivative filtering and integral clamping.

We define the velocity error as $e_k = v^*(q_k) - \hat{v}_k$. To manage oracle jitter, we implement a **Filtered Derivative**:

$$\tilde{e}_k = \alpha e_k + (1-\alpha)\tilde{e}_{k-1}, \quad D_k = K_D\frac{\tilde{e}_k - \tilde{e}_{k-1}}{\Delta t} \tag{5}$$

To prevent catastrophic overflow during sustained stress, we implement **Anti-Windup** logic:

$$\dot{I} = \begin{cases} e_k, & 0 < u_k < u_{\max} \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

The final control output u_k (the actuation signal to the rollup matching engine) is:

$$u_k = K_P e_k + K_I I_k + D_k \tag{7}$$

4 Implementation: MagicBlock SDK

The following Rust implementation highlights the bridge between the HJB strategic target and the PID tactical output.

Listing 1: Hierarchical Control Implementation

```rust
#[ephemeral_function]
pub fn step_control(ctx: Context<WickGuard>) -> Result<()> {
    // 1. Strategic Handshake (HJB)
    let v_star = kappa * ctx.accounts.position.inventory;

    // 2. Error Calculation
    let error = v_star - state.realized_velocity;

    // 3. Tactical Filtered PID
    state.filtered_error = ALPHA * error + (1.0 - ALPHA) * state.last_error;
    let d_term = KD * (state.filtered_error - state.last_filtered_error) / DT;

    // Anti-windup check
    if state.last_u < U_MAX { state.integral += KI * error * DT; }

    let u_k = KP * error + state.integral + d_term;

    // 4. Actuation
    execute_rebalance(ctx, u_k)?;
    Ok(())
}
```

5 Security and Risk Mitigation

The efficacy of a high-frequency control framework is fundamentally bounded by the quality of its inputs and the integrity of its execution environment. We identify two primary risk vectors and our corresponding mitigations.

5.1 Oracle-to-Rollup Parity

While the PID engine operates at sub-10ms intervals, it is subject to **Stale-Data Risk** if the external oracle (e.g., Pyth, Switchboard) fails to update during periods of extreme network congestion. In such scenarios, the derivative term D_k approaches zero, effectively blinding the crash-detection mechanism.

5.2 Non-Custodial Delegation

The use of an Ephemeral Rollup requires users to delegate rebalancing authority. To ensure security, WickGuard utilizes Program Derived Addresses (PDAs) with strictly scoped permissions. The "User Agent" is programmatically restricted to debt-repayment instructions; it possesses no "Withdraw" or "Transfer" authority, ensuring that collateral can only be exchanged for debt reduction within the parent lending protocol.

6 Conclusion

This framework provides a mathematically rigorous path from long-term ergodic optimality to real-time high-frequency execution. By separating strategy from tactics, WickGuard ensures that deleveraging is not only fast but economically efficient under the stated assumptions.