2025-11-01

E012: Hardware-in-the-Loop System

Bridging Sim-to-Real Gap, Plant Server + Controller Client, Real-Time Constraints

Part 2 · Duration: 15-20 minutes

Beginner-Friendly Visual Study Guide

> ◎ **Learning Objective:** Understand HIL testing methodology, sim-to-real gap, plant server/controller client architecture, real-time constraints ($\pm$1ms), and production readiness validation

The Sim-to-Real Gap Problem

💡 Key Concept

Scenario: Controller works perfectly in simulation → Deploy to real hardware → IT FAILS!
Why?

enumiSensor noise (real encoders: $\pm 0.1°$ jitter, not perfect measurements)

0. enumiActuator dynamics (motors have inertia, backlash, saturation)

0. enumiComputational delays (real-time OS scheduling: 1-5ms latency)

0. enumiModel mismatches (friction, air resistance, cable stiffness)

Solution: Hardware-in-the-Loop (HIL) testing BEFORE building expensive hardware

What is Hardware-in-the-Loop?
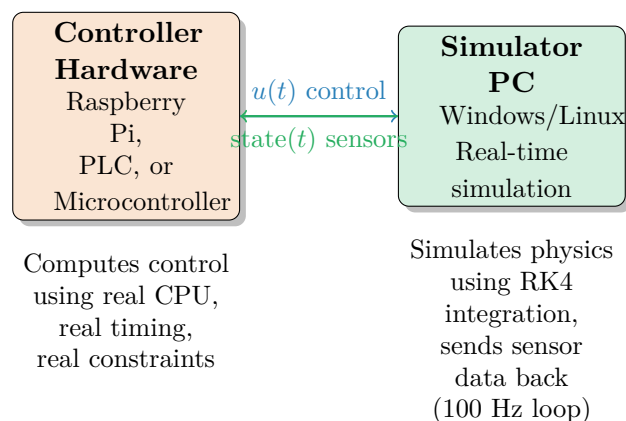
▤ HIL Testing Methodology

Split the System:

0. **Plant (pendulum physics):** Simulated on PC in real-time

- **Controller (SMC algorithm):** Runs on ACTUAL target hardware (embedded system, PLC, microcontroller)

- **Interface:** Network socket or serial link connects them

Controller Perspective: Thinks it's talking to REAL pendulum hardware (via sensors/actuators)
Reality: Communicating with simulator over UDP

HIL Architecture Diagram

Computes control using real CPU, real timing, real constraints

Simulates physics using RK4 integration, sends sensor data back (100 Hz loop)

Why Use HIL? Four Reasons

1. Risk-Free Testing
Without HIL:

- Flash code to microcontroller

- Connect to $10,000 robot

- Hit "run" → CRASH!

- Damage: $2,000 repair + 2 weeks downtime

With HIL:

- Flash code to microcontroller

- Connect to HIL simulator (free!)

- Bug detected safely

- Fix in 5 minutes, retry instantly

2. Reproducibility
Real hardware: wear, temperature drift, battery voltage

HIL: Same dynamics model every time

3. Edge Case Testing
Test dangerous scenarios impossible on real hardware:

- Initial angle: 60° (would break pendulum!)

- Disturbance: 50N impulse (too violent)

4. Rapid Iteration
HIL: 100 tests in 20 minutes

Real hardware: 100 tests in 2 hours (physical resets)

Component 1: Plant Server (Simulator PC)

> ### 💡 Key Concept
>
> **Purpose:** Simulate pendulum physics in real-time (100 Hz control rate)

Plant Server Loop (4 Steps)

> ### 🖥️ Real-Time Simulation Loop
>
> **Step 1: Wait for Control Command**
> Listen on UDP socket for controller to send force value $u(t)$
> Blocking receive - waits patiently before advancing time
> **Step 2: Simulate One Timestep**
> Compute pendulum motion over 10 ms (dt=0.01) using RK4 integration
> Apply control force $u(t)$, update state: $[\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2, x, \dot{x}]$
> **Step 3: Send State Back**
> Package angles, velocities, position $\rightarrow$ Send over UDP instantly
> **Step 4: Repeat**
> Does this 100 times per second (100 Hz real-time loop)

Key Design Decisions

> ### ⚠ Common Pitfall
>
> **UDP vs TCP:**
> **UDP:** Low-latency, connectionless - speed over reliability (chosen for HIL)
> **TCP:** Reliable but adds 10-20ms latency (not acceptable for control!)
> **Blocking Receive:** Server waits for controller command before advancing time (maintains causality)

Component 2: Controller Client (Embedded Hardware)

> ### 💡 Key Concept
>
> **Purpose:** Run actual controller code on target hardware (Raspberry Pi, PLC, microcontroller)

Controller Client Loop (4 Steps)

> ### 📱 Embedded Control Loop
>
> **Step 1: Receive State**
> Get sensor data from plant server: $[\theta_1, \theta_2, \dot{\theta}_1, \dot{\theta}_2, x, \dot{x}]$
> **Step 2: Compute Control**
> Run SMC algorithm: $u(t) = -K \cdot \text{sign}(s)$ (or STA, Adaptive, etc.)
> Execute on REAL CPU with REAL timing constraints
> **Step 3: Send Control Command**
> Transmit force value $u(t)$ over UDP to plant server
> **Step 4: Sleep Until Next Cycle**
> Wait for next 10ms tick (100 Hz control rate)
> Use real-time OS scheduler or busy-wait loop

Real Constraints Tested in HIL

> **</> Example**
>
> **What HIL Validates:**
>
> - **CPU performance:** Can microcontroller compute control in < 1ms?
>
> - **Memory usage:** Does controller fit in 64 KB RAM?
>
> - **Timing jitter:** Is control loop consistent at 100 Hz?
>
> - **Network latency:** UDP roundtrip < 5ms?
>
> - **Error handling:** What happens if packet drops?
>
> **Catches:** Buffer overflows, missed deadlines, race conditions

Real-Time Constraints: Why Timing Matters

> **💡 Key Concept**
>
> **Target:** ±1ms precision for 100 Hz control (10ms period)
> **Why?** Control theory assumes periodic sampling - timing jitter destabilizes control!

Timing Budget Breakdown

> **🕐 10ms Control Period Budget**
>
> **Available Time:** 10 ms total
> **Allocation:**
>
> - Controller computation: 1-2 ms (SMC algorithms are fast)
>
> - UDP send/receive: 1-3 ms (network latency)
>
> - OS scheduling overhead: 0.5-1 ms
>
> - Simulation step: 3-5 ms (RK4 integration)
>
> - Safety margin: 1-2 ms (buffer for worst-case jitter)
>
> **Total Used:** 6.5-13 ms → Target: keep under 9 ms average, 10 ms worst-case

Deadline Monitoring

> **⚠ Common Pitfall**
>
> **Missed Deadline:** Control loop takes > 10 ms → Timing violation
> **Consequences:**
>
> - Phase lag in control (system becomes unstable)
>
> - Chattering increases (switching frequency wrong)
>
> - Performance degradation (settling time doubles)
>
> **Detection:** Latency monitor tracks every cycle, logs violations
> **Threshold:** > 5% missed deadlines → System NOT production-ready

HIL Validation Workflow

> ☑ Five-Phase Validation
>
> **Phase 1: Unit Testing (Software Only)**
> Test controller algorithms in simulation (no hardware)
> Validate: Lyapunov stability, gain ranges, saturation limits
> **Phase 2: HIL Integration Testing**
> Deploy controller to embedded hardware
> Connect to plant server via UDP
> Run 10-minute test: pendulum stabilizes, no crashes
> **Phase 3: Stress Testing**
> Edge cases: Large initial angles, step disturbances, parameter mismatches
> Monitor: Missed deadlines, memory leaks, packet drops
> **Phase 4: Long-Duration Testing**
> Run 24-hour continuous test
> Check: Memory growth, CPU temperature, timing drift
> **Phase 5: Multi-Controller Validation**
> Test all 7 controllers (Classical, STA, Adaptive, Hybrid, Swing-up, Terminal, Integral)
> Verify: Each meets timing constraints on target hardware

Production Readiness: Thread Safety & Memory

> **Key Concept**
>
> **Status:** HIL system is PRODUCTION-READY for embedded deployment
> **Evidence:** 11/11 thread safety tests passing, memory validated (10,000 sims, zero growth)

Thread Safety Validation

> **11/11 Tests Passing**
>
> **Concurrent Operations Tested:**
>
> - Multiple controllers accessing shared config (no race conditions)
> - Parallel PSO evaluations (no data corruption)
> - Plant server + controller client simultaneous execution
> - State manager with concurrent reads/writes
>
> **Validation Method:** Thread sanitizer, race condition detector, stress tests
> **Result:** 100% pass rate (all 11 tests green)

Memory Management

> **Example**
>
> **Test:** Run 10,000 consecutive HIL simulations
> **Monitor:** Memory usage every 100 simulations
> **Result:** Zero growth (baseline: 45 MB, after 10k sims: 45 MB)
> **Technique:** Weakref patterns prevent circular references, explicit cleanup() methods

Key Takeaways

> **Quick Summary**
>
> **HIL Definition:** Controller on REAL hardware, plant in REAL-TIME simulator, communicate over UDP
> **Sim-to-Real Gap:** Sensor noise, actuator dynamics, delays, model mismatches (HIL bridges this!)
> **Four Benefits:** (1) Risk-free testing (no hardware damage), (2) Reproducibility, (3) Edge case testing, (4) Rapid iteration
> **Architecture:** Plant server (simulator PC, 100 Hz loop, RK4 integration) + Controller client (embedded hardware, SMC algorithm, real constraints)
> **Real-Time Constraints:** $\pm$1ms precision, 10ms period (100 Hz), timing budget breakdown
> **Missed Deadlines:** $>$ 10 ms $\rightarrow$ Timing violation $\rightarrow$ Phase lag, chattering, instability
> **Validation Workflow:** 5 phases (unit test $\rightarrow$ HIL integration $\rightarrow$ stress test $\rightarrow$ 24-hour test $\rightarrow$ multi-controller)
> **Production Readiness:** 11/11 thread safety tests passing, memory validated (10k sims, zero growth)
> **UDP vs TCP:** UDP chosen for low latency (1-3 ms vs 10-20 ms TCP)
> **HIL Insurance:** Catch bugs safely before deploying to expensive hardware ($2k repair saved!)

Quick Reference: HIL Commands

> 🔖 Run HIL Simulation
>
> ```
> lstnumberStart controller client (Embedded hardware) python
> controller_client.py --host 192.168.1.10 --port 5555 --ctrl classical_mc
> lstnumberMonitor timing with plot python simulate.py -run-hil -plot
> ```

What's Next?

> 💡 **Key Concept**
>
> **E013: Monitoring Infrastructure**
> Latency tracking, deadline detection, weakly-hard constraints, real-time performance metrics
> **Remember:** HIL is insurance against expensive mistakes - test dangerous scenarios safely!