

2025-11-01

# E011: Configuration & Deployment

config.yaml as Cockpit Control Panel, Pydantic Validation, 3 Deployment Scenarios

Part 2 · Duration: 15-20 minutes

*Beginner-Friendly Visual Study Guide*

🎯 **Learning Objective:** Understand config.yaml structure (6 sections, 400 lines), Pydantic validation, physics constraints, controller parameters, and 3 deployment environments

## The Configuration Problem

### 💡 Key Concept

**Question:** How do you prevent users from setting cart mass to -5 kg?

**Answer:** Validate configuration BEFORE simulation runs (not during, not after)

**Principle:** Invalid config → Program refuses to start (no silent failures!)

## Why Not Hardcode Parameters?

### ⚠️ Three Problems with Hardcoding

#### Problem 1: Reproducibility

Controller gains scattered across 10 Python files → How to reproduce experiment?

**Solution:** Single config file versioned in Git → Entire simulation reproducible

#### Problem 2: Validation

Hardcoded values can't be validated before runtime

**Solution:** Config file parsed, validated, rejected if invalid

#### Problem 3: Parameter Sweeps

Want to test 100 gain combinations → Must edit code 100 times

**Solution:** Generate 100 config files, no code changes

## config.yaml: The Cockpit Control Panel

### 💡 Key Concept

**Analogy:** Like pilot's cockpit with labeled switches for engines, flaps, landing gear

**Our Config:** 6 main sections (panels) controlling different systems

**Size:** 400 lines with extensive inline comments

## Six Main Sections

### 1. Physics Panel

- Masses (cart, pendulum 1, pendulum 2)
- Lengths (L1, L2)
- Friction coefficients
- Gravity
- Moments of inertia

### 2. Controllers Panel

- Gains for each of 7 controllers
- Boundary layers
- Adaptation rates
- Max force limits

### 3. PSO Optimization Panel

- Particle count (50)
- Iterations (100)
- Inertia weight (0.7)
- Cost function weights

**4. Simulation Panel**

- Duration (10 seconds)
- Timestep ( $0.01 \text{ s} = 100 \text{ Hz}$ )
- Initial conditions
- Dynamics model (simplified/full)

**5. Hardware-in-the-Loop Panel**

- Network settings

- IP addresses

- Timeouts

**6. Monitoring Panel**

- Latency thresholds
- Deadline detection
- Logging verbosity

## Physics Parameters: Defining the System

### 12 Physical Parameters

**Masses:** Cart, pendulum 1, pendulum 2

**Lengths:** L1, L2 (pendulum link lengths)

**Centers of Mass:** Distance from pivot to center

**Moments of Inertia:** Rotational resistance

**Environment:** Gravity (9.81 m/s<sup>2</sup>), friction coefficients

**Defines:** Whether simulating lightweight lab prototype or heavy industrial system

## Physics Validation: Preventing Impossible Configurations

### Common Pitfall

**Example 1: Negative Mass**

**Config:** cart\_mass: -5.0

**Problem:** Newton's  $F=ma$  with negative mass  $\rightarrow$  Accelerates TOWARD you when pushed away (universe doesn't work this way!)

**Pydantic Error:** "Field 'cart\_mass' expected positive value, received -5.0. Physical masses cannot be negative."

**Result:** Simulation refuses to start, user fixes config

### Common Pitfall

**Example 2: Impossible Inertia**

**Physics Constraint:**  $\text{Inertia} \geq \text{mass} \times \text{length}^2$

**Config:** pendulum1\_inertia: 0.0001 (but mass=0.2 kg, length=0.4 m  $\rightarrow$  min inertia=0.008)

**Validation Error:** "pendulum1\_inertia violates physics constraints. You configured 0.0001, but minimum for 0.2 kg at 0.4 m is 0.008."

**Prevents:** Hours wasted running simulation with meaningless results

## Controller Configuration: Seven Variants

### Example

**Example: Classical SMC Config**

```
lstnumber controllers:
lstnumber   classical_smc:
lstnumber     gains: [23.07, 12.85, 5.51, 3.49, 2.23, 0.15]
lstnumber     max_force: 150.0
lstnumber     dt: 0.001
lstnumber     boundary_layer: 0.3
```

**Gains Array:** 6 values (sliding surface coefficients + switching gains)

**Boundary Layer:** 0 = perfect but chattering, 0.3 = smooth in 0.3-rad band

## Controller-Specific Gain Validation

### ✓ Schema Enforcement

#### Pydantic Schema Per Controller:

- ClassicalSMCConfig expects EXACTLY 6 gains
- STASMCConfig expects EXACTLY 6 gains (different interpretation)
- AdaptiveSMCConfig expects EXACTLY 5 gains

**Wrong Number?** "classical\_smc.gains expected 6 elements, received 5"

**Prevents:** Mismatched gain arrays causing runtime crashes

## PSO Configuration: Optimization Parameters

### ⚙️ PSO Algorithm Parameters

**n\_particles: 50** - How many candidate solutions search simultaneously

**n\_iterations: 100** - How many generations swarm evolves

**inertia\_weight: 0.7** - Exploration vs exploitation tradeoff

**cognitive\_coeff: 2.0** - Trust own best position

**social\_coeff: 2.0** - Trust swarm's best position

**cost\_weights:** Relative importance in cost function

- **state\_error:** 1.0 (minimize tracking error most)
- **control\_effort:** 0.1 (care less about actuator wear)
- **settling\_time:** 0.5 (penalize slow settling moderately)
- **overshoot:** 0.3 (penalize overshoot somewhat)

### 💡 Pro Tip

**Tuning Cost Weights:**

**Industrial Robot:** control\_effort: 0.5 (actuators expensive, wear out)

**Academic Benchmark:** state\_error: 2.0 (maximize tracking accuracy)

**No Universal Right Answer** - depends on application priorities

## Simulation Settings

**duration: 10.0 seconds**

How long to simulate

**dt: 0.01 seconds (100 Hz)**

Timestep for numerical integration

**initial\_state:**

[0.0, 0.05, -0.03, 0.0, 0.0, 0.0]

Cart at origin, small angle perturbation

**use\_full\_dynamics: false**

Simplified (5  $\mu$ s) vs Full nonlinear (50  $\mu$ s)

**Tradeoff:**

- Simplified: PSO in 3 minutes
- Full: PSO in 30 minutes
- Strategy: Develop with simplified, validate with full

## Timestep Constraints: Nyquist Stability

### ⚠️ Common Pitfall

**Physics Limit:** Fastest dynamics 5 rad/s (pendulum natural frequency)

**Nyquist:** Need 10 samples per oscillation  $\rightarrow$  dt 0.1 seconds

**Practice:** dt=0.01 for 100 Hz (10 $\times$  safety margin)

**Validation:** If dt=0.5  $\rightarrow$  Warning: "Timestep 0.5s exceeds Nyquist limit for system dynamics. Maximum safe: 0.1s"

## Pydantic Validation: The Bouncer

### 💡 Key Concept

**Analogy:** Pydantic is a nightclub bouncer checking IDs at the door

**Checks:**

enumiData types correct? (Is cart\_mass a number or "heavy"?)

0. enumiValues in acceptable ranges? (Mass positive? Length < 5 m?)

**Pass:** Simulation starts

**Fail:** Rejected with specific reason (before wasting time!)

### Fail Fast, Fail Clearly

### 🔗 Example

**Without Bouncer:**

0. Start simulation
  - Run for 3 minutes
  - Crash deep in physics engine: "cannot multiply string by float"
  - Debug backwards: what config value was wrong?

**With Pydantic Bouncer:**

- Try to load config
- Rejected at door immediately: "cart\_mass must be float, you provided string"
- Fix config, retry - no time wasted!



## Configuration Loading Process

### 📄 Five-Step Process

- Step 1:** Read `config.yaml` from disk (PyYAML → nested dictionary)
- Step 2:** Pass to Pydantic `Config` model (recursively validate all sections)
- Step 3:** Additional physics validation (inertia bounds, Nyquist criterion)
- Step 4:** Cross-section validation (if `use_full_dynamics=true`, check consistency)
- Step 5:** If all pass → Return immutable config object. If any fail → Collect ALL errors, report together

### 💡 Pro Tip

#### Why Collect All Errors?

**Bad UX:** Fix one error → Re-run → Hit next error → Fix → Hit third error (frustrating!)

**Good UX:** See all errors at once → "cart\_mass negative, pendulum1\_inertia violates physics, classical\_smc.gains wrong length (5 expected 6)" → Fix all in batch

## Deployment: Three Environments

### 🚀 Deployment Scenarios

#### Environment 1: Local Development

- Clone repo, create venv, `pip install -r requirements.txt`
- Run `python simulate.py`
- Everything on one machine, default config, immediate feedback

#### Environment 2: Batch Computation

- Research cluster or cloud instance
- Parameter sweeps, Monte Carlo simulations, PSO optimization
- Headless execution (no GUI), results saved to files

#### Environment 3: Hardware-in-the-Loop (HIL)

- Plant server on one machine (PLC or industrial PC with real hardware)
- Controller client on another machine
- Communicate over network using ZeroMQ

## Key Takeaways

### ☰ Quick Summary

**config.yaml:** 400 lines, 6 sections (physics, controllers, PSO, simulation, HIL, monitoring)

**Cockpit Control Panel:** Every parameter has inline comment explaining purpose, units, citations

**Physics Validation:** Prevents negative mass, impossible inertia, violates Nyquist limits

**Controller Gains:** Schema per controller (Classical=6, STA=6, Adaptive=5 gains)

**Boundary Layer:** 0 = perfect/chattering, 0.3 = smooth (chattering mitigation)

**PSO Parameters:** 50 particles, 100 iterations, cost weights tunable per application

**Simulation:** `dt=0.01` (100 Hz), `duration=10s`, simplified vs full dynamics tradeoff

**Pydantic Bouncer:** Validates types + ranges BEFORE simulation (fail fast, fail clearly)

**Loading Process:** Read YAML → Pydantic validate → Physics check → Cross-section → Immutable config or detailed errors

**Three Environments:** Local dev (one machine), Batch compute (headless), HIL (networked plant + controller)

**Configuration as Code:** Version in Git for reproducibility, generate variants for parameter sweeps

## Quick Reference: Config Usage

### 📖 Load and Validate Config

```
lstnumberLoad with validation config = load_config("config.yaml")
lstnumberStrict mode (fail on unknown keys) config = load_config("config.yaml", allow_unknown = False)
lstnumberAccess nested values
```

## What's Next?

### 💡 Key Concept

#### **E012: Hardware-in-the-Loop System**

Plant server, controller client, network communication, real-time constraints, latency measurement

**Remember:** Configuration is a first-class artifact, not an afterthought!