

2025-11-01

E030: Controller Base Classes & Factory

Understanding the Foundation: How
5 Controllers Share One Interface

Part 5A · Duration: 25-30 minutes

Beginner-Friendly Visual Study Guide

🎯 **Learning Objective:** Understand the abstract base class that all controllers inherit from, the factory pattern for creating controllers, and how this enables seamless controller swapping

The Design Challenge

💡 Key Concept

One Interface, Five Controllers: All controllers (Classical SMC, STA, Adaptive, Hybrid Adaptive STA, Conditional Hybrid) implement the SAME interface. The factory pattern makes adding new controllers (e.g., Swing-Up, MPC) trivial - just register them!

Result: Change one line in config.yaml and swap algorithms without touching code!

Why This Matters

⚠️ Common Pitfall

Without Interface:

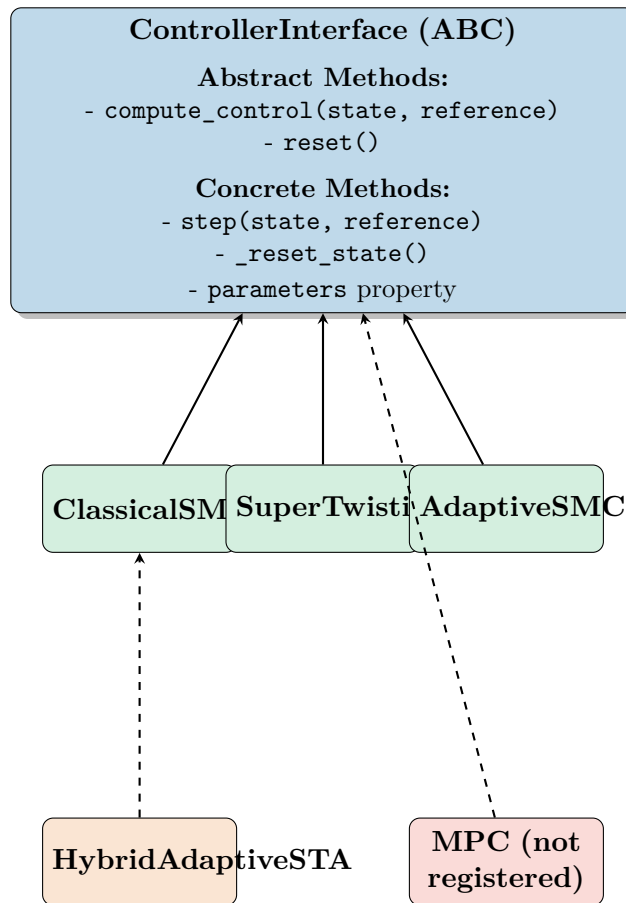
- Each controller has different method names
- Simulation code full of if/else chains
- Adding new controller = rewrite everything
- Testing = nightmare (5 different APIs)

💡 Pro Tip

With Interface:

- One method: `compute_control()`
- Simulation code agnostic to controller type
- New controller = implement interface
- Testing = same harness for all 5

The ControllerInterface Abstract Base Class



Example

Python ABC Pattern: Abstract Base Class (ABC) defines a contract. Subclasses MUST implement abstract methods or Python raises `TypeError` at instantiation.

Core Interface Code (src/controllers/base/controller_interface.py)

```

from abc import ABC, abstractmethod
from typing import Optional, Tuple, Any
import numpy as np

class ControllerInterface(ABC):
    """Abstract base class for all controllers in the DIP system."""

    def __init__(self, max_force: float = 20.0, dt: float = 0.01):
        """Initialize base controller with common parameters."""
        self.max_force = max_force  # Actuator saturation limit (N)
        self.dt = dt  # Sampling timestep (s)
        self._reset_state()

    @abstractmethod
    def compute_control(self, state: np.ndarray,
                       reference: Optional[np.ndarray] = None) -> float:
        """THE KEY METHOD - Compute control force for given state.

        Args:
            state: [x, xdot, theta1, thetadot, theta2, theta2dot]
            reference: Target state (default: upright equilibrium)

        Returns:
            float: Control force to apply to cart (N)
        """
        pass  # Subclasses MUST implement

    @abstractmethod
    def reset(self) -> None:
        """Reset controller internal state (for multi-simulation)."""
        pass

    def step(self, state: np.ndarray,
             reference: Optional[np.ndarray] = None) -> Tuple[float, Any]:
        """Perform one control step with saturation.

        control = self.compute_control(state, reference)

        # Apply actuator limits (CRITICAL for real hardware!)
        control = np.clip(control, -self.max_force, self.max_force)

        # Return control + diagnostics
        info = {'saturated': bool(abs(control) >= self.max_force),
  
```

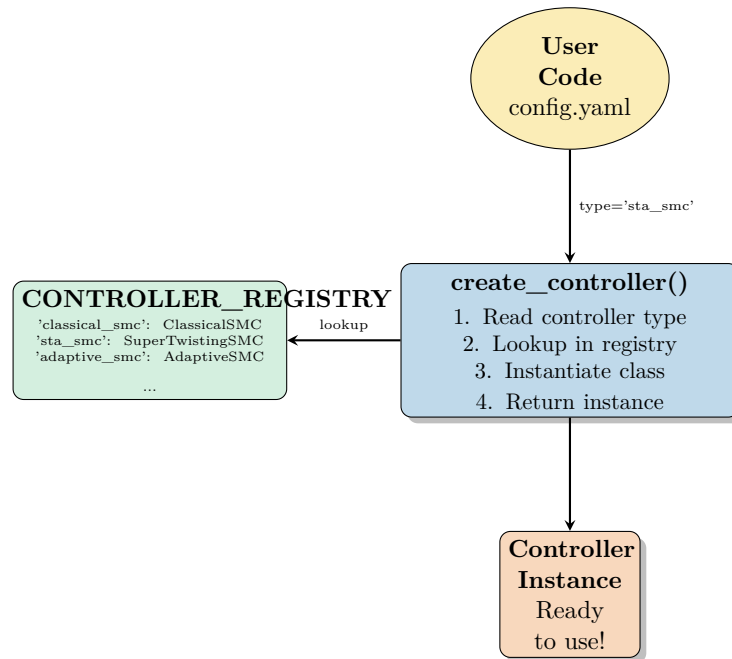
```
lstnu0er         'control_raw': control}
lstnu0er         return control, info
lstnu0er
lstnu0er     @property
lstnu0er     def parameters(self) -> dict:
lstnu0er         """Get controller parameters for logging/analysis."""
lstnu0er         return {'max_force': self.max_force, 'dt': self.dt}
```

Factory Pattern: One Function to Rule Them All

💡 Key Concept

Problem: How do you create 5 controller types from configuration without massive if/else chains?
Solution: Factory pattern - one function `create_controller()` that uses a registry to instantiate the right class.

Factory Pattern Flow



Factory Code (src/controllers/factory/base.py)

```

lstnu0er# Simplified version for clarity (verified 2026-02-04)
lstnu0erCONTROLLER_REGISTRY = {
lstnu0er    'classical_smc': ClassicalSMC,
lstnu0er    'sta_smc': SuperTwistingSMC,
lstnu0er    'adaptive_smc': AdaptiveSMC,
lstnu0er    'hybrid_adaptive_sta_smc': HybridAdaptiveSTASMC,
lstnu0er    'conditional_hybrid': ConditionalHybrid,
lstnu0er    # Note: SwingUp and MPC not yet registered (can be added by extending registry)
lstnu0er}
lstnu0er
lstnu0erdef create_controller(ctrl_type: str, config: dict, gains: list) -> ControllerInterface:
lstnu0er    """Factory function to create any controller type.
lstnu0er
lstnu0er    Args:
lstnu0er        ctrl_type: Controller identifier (e.g., 'sta_smc')
lstnu0er        config: Configuration dictionary
lstnu0er        gains: Controller gains (validated before instantiation)
lstnu0er
lstnu0er    Returns:
lstnu0er        Controller instance implementing ControllerInterface
lstnu0er
lstnu0er    Raises:
lstnu0er        ValueError: Unknown controller type
lstnu0er    """
lstnu0er    # Canonicalize type (handle aliases)
lstnu0er    ctrl_type = canonicalize_controller_type(ctrl_type)
lstnu0er
lstnu0er    # Lookup controller class in registry
lstnu0er    if ctrl_type not in CONTROLLER_REGISTRY:
lstnu0er        raise ValueError(f"Unknown controller: {ctrl_type}")
lstnu0er
lstnu0er    controller_class = CONTROLLER_REGISTRY[ctrl_type]
lstnu0er
lstnu0er    # Instantiate controller with validated parameters
lstnu0er    return controller_class(gains=gains, **config)
  
```

Registry Benefits

- **No if/else chains:** Dictionary lookup = $O(1)$
- **Easy to extend:** Add new controller = register it
- **Type-safe:** All values implement Controller-Interface

- **Discoverable:** List available controllers programmatically

Alias Support

```

lstdictCONTROLLER_ALIASES = {
lstdict    'classical': 'classical_smc',
lstdict    'sta': 'sta_smc',
lstdict    'super_twisting': 'sta_smc',
lstdict    'adaptive': 'adaptive_smc',
lstdict    # User-friendly names
lstdict}

```

Usage Example: Swapping Controllers

Python Usage

```

lstdictLoad configuration config = load_config("config.yaml")
lstdictCreate controller (type from config, NOT hardcoded!) controller = create_controller(ctrl_type=config[
lstdict    'controller_type'], 'sta_smc', config=config['controller_params'], gains=config['controller_gains'])
lstdictSimulation loop - controller type doesn't matter here! for t in np.arange(0, 10, dt): state =
lstdict    get_current_state() control = controller.compute_control(state) apply_control(control)
lstdictWant to test different algorithm? Change ONE line in config.yaml!

```

Memory Management: Breaking Circular References

⚠ Common Pitfall

The Circular Reference Problem:

Controller → holds reference to → Dynamics Model

Dynamics Model → sometimes holds reference to → Controller

Result: Python garbage collector can't free memory (memory leak!)

Solution: Weakref Pattern

Bad (Strong Reference):

```
lstnu0erclass ClassicalSMC:
lstnu0er    def __init__(self, dynamics_model):
lstnu0er        # Strong reference
lstnu0er        self.dyn = dynamics_model
```

Problem: If `dynamics_model` holds controller, both objects never freed!

Good (Weak Reference):

```
lstnu0erimport weakref
lstnu0er
lstnu0erclass ClassicalSMC:
lstnu0er    def __init__(self, dynamics_model):
lstnu0er        # Weak reference
lstnu0er        self._dynamics_ref = weakref.ref(dynamics_model)
lstnu0er
lstnu0er    @property
lstnu0er    def dyn(self):
lstnu0er        return self._dynamics_ref()
```

Weak reference doesn't prevent garbage collection!

Cleanup Pattern

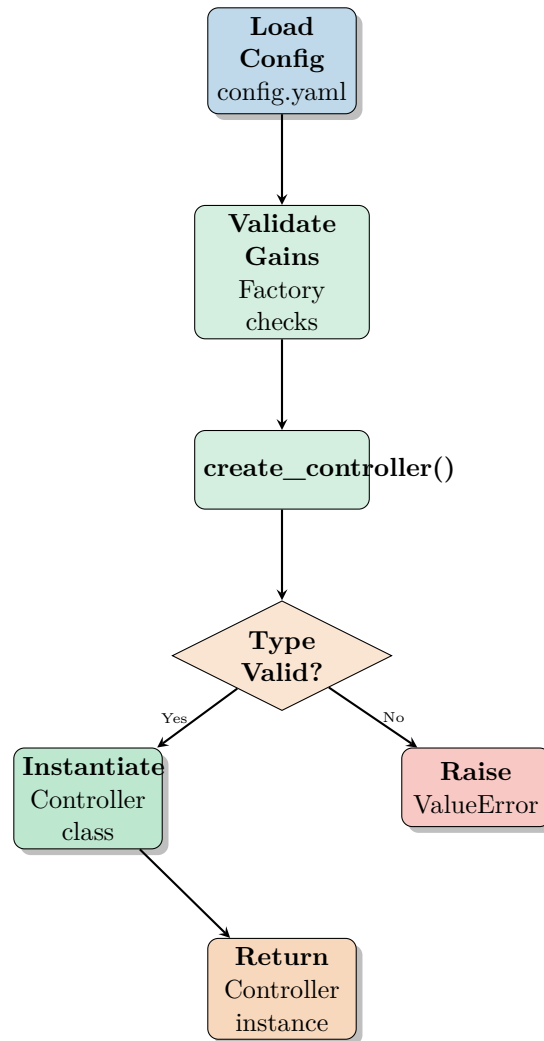
```
lstnu0erclass ClassicalSMC(ControllerInterface):
lstnu0er    def cleanup(self) -> None:
lstnu0er        """Explicit memory cleanup to prevent leaks."""
lstnu0er        # Nullify dynamics reference
lstnu0er        if hasattr(self, '_dynamics_ref'):
lstnu0er            self._dynamics_ref = lambda: None
lstnu0er
lstnu0er        # Clear large NumPy arrays
lstnu0er        if hasattr(self, 'L'):
lstnu0er            self.L = None
lstnu0er        if hasattr(self, 'B'):
lstnu0er            self.B = None
lstnu0er
lstnu0er    def __del__(self) -> None:
lstnu0er        """Destructor for automatic cleanup."""
lstnu0er        try:
lstnu0er            self.cleanup()
lstnu0er        except Exception:
lstnu0er            pass # Prevent exceptions during finalization
```

💡 Pro Tip

Memory Management Guideline: Call `cleanup()` when done with controller, especially in batch simulations or PSO optimization (1000s of instantiations).

The `__del__()` destructor provides automatic cleanup, but explicit `cleanup()` is more reliable.

Controller Initialization Flow



Design Patterns Identified

Four Key Patterns

- enumi**Abstract Base Class (ABC)**: Enforces contract via Python's `@abstractmethod`
- 0. enumi**Factory Pattern**: Registry-based instantiation decouples creation from usage
- 0. enumi**Weak Reference Pattern**: Prevents memory leaks from circular references
- 0. enumi**Strategy Pattern**: Controllers are interchangeable strategies for same problem

Practical Examples: Using the Factory

Example 1: Command-Line Simulation

```
lstnu0er# File: simulate.py
lstnu0erimport argparse
lstnu0erfrom src.controllers.factory import create_controller
lstnu0erfrom src.config import load_config
lstnu0er
lstnu0erdef main():
lstnu0er    parser = argparse.ArgumentParser()
lstnu0er    parser.add_argument('--ctrl', default='classical_smc',
lstnu0er                        help='Controller type')
lstnu0er    args = parser.parse_args()
lstnu0er
lstnu0er    # Load config
lstnu0er    config = load_config("config.yaml")
lstnu0er
lstnu0er    # Create controller from command-line argument
lstnu0er    controller = create_controller(
lstnu0er        ctrl_type=args.ctrl, # User-specified type!
lstnu0er        config=config['controllers'][args.ctrl],
lstnu0er        gains=config['gains'][args.ctrl]
lstnu0er    )
lstnu0er
lstnu0er    # Run simulation (same code for ALL controllers)
lstnu0er    results = simulate(controller, initial_state, dt=0.01, duration=10.0)
lstnu0er    plot_results(results)
lstnu0er
lstnu0er# Usage:
lstnu0er# python simulate.py --ctrl classical_smc
lstnu0er# python simulate.py --ctrl sta_smc
lstnu0er# python simulate.py --ctrl adaptive_smc
```

Example 2: Batch Comparison

```
lstnu0erfrom src.controllers.factory import list_available_controllers, create_controller
lstnu0er
lstnu0er# Discover all available controllers programmatically
lstnu0erall_controllers = list_available_controllers()
lstnu0er# Returns: ['classical_smc', 'sta_smc', 'adaptive_smc', ...]
lstnu0er
lstnu0erresults = {}
lstnu0erfor ctrl_type in all_controllers:
lstnu0er    # Create controller
lstnu0er    controller = create_controller(ctrl_type, config, default_gains[ctrl_type])
lstnu0er
lstnu0er    # Run simulation
lstnu0er    metrics = run_simulation(controller)
lstnu0er    results[ctrl_type] = metrics
lstnu0er
lstnu0er    # Clean up memory (IMPORTANT for batch!)
lstnu0er    controller.cleanup()
lstnu0er
lstnu0er# Compare all controllers
lstnu0erplot_comparison(results)
```

Example 3: PSO Optimization

```
lstnu0erfrom src.controllers.factory import create_smc_for_pso, get_gain_bounds_for_pso
lstnu0er
lstnu0erdef objective_function(gains):
lstnu0er    """PSO evaluates this function 1000s of times."""
lstnu0er    # Create controller with candidate gains
lstnu0er    controller = create_smc_for_pso('sta_smc', gains, max_force=20.0, dt=0.01)
lstnu0er
lstnu0er    # Simulate
lstnu0er    cost = simulate_and_evaluate(controller)
lstnu0er
lstnu0er    # Clean up (prevents memory leak over 1000 iterations!)
lstnu0er    controller.cleanup()
lstnu0er
lstnu0er    return cost
lstnu0er
lstnu0er# Get valid gain bounds for chosen controller type
lstnu0erbounds = get_gain_bounds_for_pso('sta_smc') # Returns: [(K1_min, K1_max), (K2_min, K2_max), ...]
lstnu0er
lstnu0er# Run PSO
lstnu0erbest_gains = pso_optimize(objective_function, bounds, n_particles=30, n_iterations=50)
```

Quick Reference: Factory API

Factory Functions

```

lstnumberDiscovery functions list_available_controllers() -> list[str] list_all_controllers() -> dictWithMetadata
lstnumberValidation validate_controller_gains(ctrl_type, gains) -> ValidationResult get_default_gains(ctrl_type) ->
    list[float] get_gain_bounds(ctrl_type) -> list[tuple]
lstnumberPSO integration
    create_smc_for_pso(ctrl_type, gains, **params) -> ControllerInterface get_gain_bounds_for_pso(ctrl_type) -> list[tuple]
lstnumberType utilities canonicalize_controller_type(name) -> str Resolves aliases

```

Configuration Example (config.yaml)

```

lstnumber# Controller selection (actual config.yaml structure verified 2026-02-04)
lstnumber# No single controller_type field - specify via simulate.py --ctrl flag
lstnumber# Controller-specific parameters
lstnumbercontrollers:
lstnumber  classical_smc:
lstnumber    max_force: 20.0
lstnumber    boundary_layer: 0.1
lstnumber    switch_method: 'tanh'
lstnumber
lstnumber  sta_smc:
lstnumber    max_force: 20.0
lstnumber    dt: 0.01
lstnumber    boundary_layer: 0.01
lstnumber    damping_gain: 0.5
lstnumber
lstnumber  adaptive_smc:
lstnumber    max_force: 20.0
lstnumber    leak_rate: 0.1
lstnumber    K_min: 1.0
lstnumber    K_max: 50.0
lstnumber    dead_zone: 0.05
lstnumber
lstnumber# Controller gains (tuned via PSO or manually)
lstnumbergains:
lstnumber  classical_smc: [23.068, 12.854, 5.515, 3.487, 2.233, 0.148] # MT-8 robust optimized gains
lstnumber  sta_smc: [15.0, 10.0, 5.0, 3.0, 2.0, 1.0] # [K1, K2, k1, k2, lam1, lam2]
lstnumber  adaptive_smc: [8.0, 4.0, 6.0, 2.5, 0.8] # [k1, k2, lam1, lam2, gamma]

```

Key Takeaways

Quick Summary

- Interface Unity:** `ControllerInterface` enforces contract - all 5 controllers implement `compute_control()`
- Factory Power:** One function creates any controller via registry pattern - no if/else chains!
- Memory Safety:** Weakref pattern prevents circular reference leaks (critical for batch simulations)
- Configuration-Driven:** Change controller algorithm by editing ONE line in `config.yaml`
- Discoverable:** Programmatically list/validate controllers for testing and optimization

What's Next?

Key Concept

E031: Classical SMC Implementation - Deep-dive into the baseline algorithm: sliding surface, switching control, equivalent control, and the chattering phenomenon

E032: Super-Twisting Algorithm (STA) - 2nd-order sliding mode for smooth control without chattering

E033-E036: Adaptive controllers, Swing-Up, MPC, and testing strategies

Code References

- 0. `src/controllers/base/controller_interface.py:12-101` - Base class definition
- `src/controllers/factory/base.py:25-90` - Factory function
- `src/controllers/factory/registry.py:10-60` - Controller registry
- `src/controllers/smc/classic_smc.py:187-190` - Weakref example