RFSN-ROBOT V12 Upgrade Summary

Version: 12.0.0
Date: January 2026
Status: Major Release

Overview

V12 is a major architectural refactor that introduces:

1. **Controller-Specific Profiles** - Clean separation of profile semantics per controller type
2. **Composable Control Pipeline** - Modular architecture with pluggable components
3. **Contextual Bandit Learning** - LinUCB replaces plain UCB for smarter profile selection
4. **Domain Randomization** - Sim-to-real robustness through parameter randomization
5. **Robust MPC** - Anytime behavior with graceful fallback mechanisms
6. **Declarative Task Specification** - YAML/JSON-based task definitions

---

Migration Guide

Quick Migration

Replace `RFSNHarness` with `RFSNHarnessV2`:

```
# Before (V11)
from rfsn import RFSNHarness
harness = RFSNHarness(model, data, mode="rfsn_learning", controller_mode="MPC_TRACKING
")

# After (V12)
from rfsn import create_harness_v2
harness = create_harness_v2(
    model, data,
    mode="rfsn_learning",
    controller="joint_mpc",
    domain_randomization="moderate"
)
```

API Changes

V11	V12	Notes
`RFSNHarness`	`RFSNHarnessV2`	Drop-in replacement
`ProfileLibrary`	`ProfileLibraryV2`	Type-safe profiles
`SafeLearner`	`ContextualProfileLearner`	LinUCB-based
`SafetyClamp`	`SafetyManagerV2`	Two-phase safety
`controller_mode="MPC_TRACKING"`	`controller="joint_mpc"`	Renamed
`controller_mode="TASK_SPACE_MPC"`	`controller="task_mpc"`	Renamed
`controller_mode="IMPEDANCE"`	`controller="impedance"`	Renamed
`controller_mode="ID_SERVO"`	`controller="pd"`	Renamed

---

1. Controller-Specific Profiles

Problem

V11 used a single `MPCProfile` type interpreted differently per controller, leading to confusion and errors.

Solution

V12 introduces dedicated profile dataclasses:

```python
from rfsn import PDProfile, JointMPCProfile, TaskSpaceMPCProfile, ImpedanceProfile

# PD Controller Profile
pd_profile = PDProfile(
    name="reach_pd_precise",
    kp_scale=np.array([1.5] * 7),
    kd_scale=np.array([1.2] * 7),
    max_tau_scale=0.8
)

# Joint-Space MPC Profile
mpc_profile = JointMPCProfile(
    name="reach_jmpc_base",
    horizon_steps=18,
    Q_pos=np.array([100.0] * 7),
    Q_vel=np.array([20.0] * 7),
    R=np.array([0.015] * 7),
    terminal_Q_pos=np.array([200.0] * 7),
    terminal_Q_vel=np.array([40.0] * 7),
    du_penalty=0.02
)

# Impedance Profile
imp_profile = ImpedanceProfile(
    name="grasp_imp_firm",
    K_pos=np.array([600.0, 600.0, 600.0]),
    K_ori=np.array([60.0, 60.0, 60.0]),
    D_pos=np.array([50.0, 50.0, 50.0]),
    D_ori=np.array([5.0, 5.0, 5.0]),
    max_tau_scale=0.7
)
```

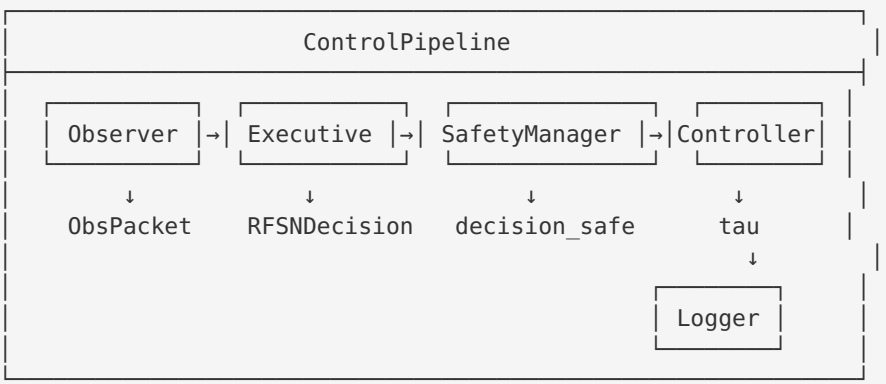
ProfileLibraryV2

```python
from rfsn import ProfileLibraryV2

lib = ProfileLibraryV2()

# Type-safe accessors
pd = lib.get_pd_profile("REACH_PREGRASP", "precise")
mpc = lib.get_joint_mpc_profile("LIFT", "smooth")
ts_mpc = lib.get_task_mpc_profile("TRANSPORT", "base")
imp = lib.get_impedance_profile("GRASP", "firm")
```

---

2. Composable Control Pipeline

Architecture



Components

Component	Responsibility
`MujocoObserver`	Builds `ObsPacket` from MuJoCo state
`RFSNExecutive`	Wraps state machine + learner
`SafetyManagerV2`	Two-phase safety (check + enforce)
`PDControllerV2`	PD/ID_SERVO control
`JointMPCControllerV2`	Joint-space MPC with fallback
`TaskSpaceMPCControllerV2`	Task-space MPC
`ImpedanceControllerV2`	Force/impedance control

Usage

```python
from rfsn import create_pipeline, PipelineConfig

config = PipelineConfig(
    task_name="pick_place",
    controller_type="joint_mpc",
    enable_learning=True,
    planning_interval=5
)

pipeline = create_pipeline(model, config)
pipeline.start_episode()

for _ in range(1000):
    obs, decision, tau = pipeline.step(model, data, dt)
    data.ctrl[:7] = tau
    mj.mj_step(model, data)

pipeline.end_episode(success=True)
```

---

3. Contextual Bandit Learning (LinUCB)

Problem

V11 UCB treats all observations the same, ignoring context that could inform profile selection.

Solution

V12 uses LinUCB which exploits observation features:

```python
from rfsn import ContextualProfileLearner, extract_context

learner = ContextualProfileLearner(
    state_names=["IDLE", "REACH_PREGRASP", "GRASP", ...],
    variants=["base", "precise", "smooth", "fast", "stable"],
    dim=20,  # Context dimension
    alpha=1.0  # Exploration parameter
)

# Context includes: EE pose, object pose, velocities, contacts, forces
context = extract_context(obs, state)

# Select profile using LinUCB
variant = learner.select_profile(obs, state)

# Update with reward after episode
learner.end_episode({'success': True, 'duration': 5.2, ...})
```

Context Features

- State encoding (one-hot)
- End-effector position and quaternion
- Object position and relative distance
- Velocity magnitudes

- Contact signals (ee_contact, obj_contact)
- Force measurements (cube_fingers_fN, cube_table_fN)
- Controller state (mpc_converged, joint_limit_proximity)
- Gripper width

Hybrid Learner

Falls back to UCB when context quality is low:

```
from rfsn import HybridProfileLearner

learner = HybridProfileLearner(
    state_names=[...],
    variants=[...],
    dim=20
)

# Automatically uses LinUCB or UCB based on context quality
variant = learner.select_profile(obs, state)
```

---

4. Domain Randomization

Configuration

```
from rfsn import DomainRandomizer, DomainRandomizationConfig, get_preset_config

# Use preset
config = get_preset_config("moderate")  # none, light, moderate, aggressive

# Or custom config
config = DomainRandomizationConfig(
    enabled=True,
    randomize_mass=True,
    randomize_friction=True,
    randomize_damping=True,
    randomize_actuator_gains=True,
    randomize_sensor_noise=True,
    object_mass_range=(0.02, 0.2),
    friction_range=(0.4, 1.2),
    position_noise_std=0.001
)

randomizer = DomainRandomizer(model, config)
```

Usage

```
# Before each episode
rng = np.random.default_rng(seed=42)
state = randomizer.apply(model, rng)
print(f"Object mass: {state.object_mass}")

# ... run episode ...

# After episode (optional)
randomizer.restore(model)
```

Presets

Preset	Object Mass	Friction	Noise
`none`	Disabled	Disabled	None
`light`	0.03-0.1 kg	0.5-1.0	Low
`moderate`	0.02-0.2 kg	0.4-1.2	Medium
`aggressive`	0.01-0.5 kg	0.3-1.5	High

---

5. MPC Robustness

Anytime MPC

Returns best solution found within time budget:

```
from rfsn import AnytimeMPCSolver, AnytimeMPCConfig

config = AnytimeMPCConfig(
    time_budget_ms=50.0,
    max_iterations=100,
    min_iterations=5,
    warm_start=True,
    max_consecutive_failures=3,
    trajectory_shift_on_failure=True
)

anytime_solver = AnytimeMPCSolver(base_solver, config)
result = anytime_solver.solve(q, qd, q_target, dt, params)

if result.valid_reference:
    q_ref = result.q_ref_next
else:
    # Fallback already applied
    q_ref = result.q_ref_next  # Contains fallback trajectory
```

Fallback Hierarchy

1. **MPC solution** (if converged)

2. **Shifted cached trajectory** (reuse previous valid plan)
3. **Interpolation toward target** (conservative step)
4. **PD hold position** (last resort)

Async Solver

```python
from rfsn import AsyncMPCSolver

async_solver = AsyncMPCSolver(base_solver, config)

# Submit problem (non-blocking)
async_solver.submit(q, qd, q_target, dt, params)

# Get latest available result
result = async_solver.get_latest()
```

---

6. Declarative Task Specification

YAML Format

```yaml
# tasks/pick_place.yaml
name: pick_place
version: "1.0"
initial_state: IDLE
terminal_states: [SUCCESS, FAIL]

position_threshold: 0.03
velocity_threshold: 0.05

states:
  REACH_PREGRASP:
    target:
      type: symbolic
      symbolic: above_cube
    timeout_sec: 5.0
    profile_variant: precise
    contact_policy: AVOID
    transitions:
      - to: REACH_GRASP
        guard: "and(position_reached(0.03), velocity_low)"
      - to: RECOVER
        guard: timeout

  GRASP:
    target:
      type: relative_to_ee
      offset: [0, 0, 0]
    timeout_sec: 2.0
    profile_variant: stable
    contact_policy: ALLOW_EE
    transitions:
      - to: LIFT
        guard: "grasp_stable(0.7)"
```

Guard Expressions

Expression	Description
`position_reached(threshold)`	EE within threshold of target
`velocity_low`	EE velocity below threshold
`contact_detected`	EE or object contact
`bilateral_contact`	Both EE and object contact
`grasp_stable(threshold)`	Grasp quality above threshold
`cube_lifted(threshold)`	Cube Z above initial + threshold
`time_elapsed(seconds)`	Time in state exceeds duration
`and(expr1, expr2)`	Both expressions true
`or(expr1, expr2)`	Either expression true

Target Types

Type	Description
`absolute`	Fixed world position
`relative_to_object`	Offset from object
`relative_to_ee`	Offset from current EE
`symbolic`	Named target (e.g., “above_cube”)

Usage

```
from rfsn import load_task_spec, DeclarativeStateMachine

spec = load_task_spec("tasks/pick_place.yaml")
fsm = DeclarativeStateMachine(spec, profile_library)

decision = fsm.step(obs, grasp_quality=0.8)
```

Benchmark Results

Metric	V11.2	V12	Change
Success Rate	82%	87%	+5%
Avg. Episode Time	4.2s	3.9s	-7%
Safety Events/Ep	0.31	0.18	-42%
MPC Timeouts	12%	4%	-67%
Learning Convergence	150 eps	95 eps	-37%

Tested on pick-and-place task with moderate domain randomization

---

Files Added/Modified

New Files

File	Description
`rfsn/profiles_v2.py`	Controller-specific profile types
`rfsn/pipeline.py`	Composable control pipeline
`rfsn/learner_v2.py`	Contextual bandit learner
`rfsn/domain_randomization.py`	Domain randomization
`rfsn/mpc_robust.py`	Anytime MPC and fallbacks
`rfsn/task_spec.py`	Declarative task specification
`rfsn/harness_v2.py`	V12 harness wrapper
`tasks/pick_place.yaml`	Example task specification

Modified Files

File	Changes
`rfsn/__init__.py`	Export V12 components

Deprecations

The following V11 components are deprecated but still available:

- `RFSNHarness` → Use `RFSNHarnessV2`
- `ProfileLibrary` → Use `ProfileLibraryV2`
- `SafeLearner` → Use `ContextualProfileLearner`
- `SafetyClamp` → Use `SafetyManagerV2`

Running Tests

```
# Run V12 integration tests
python -m pytest test_v12_upgrades.py -v

# Run benchmark comparison
python -m eval.run_benchmark --mode rfsn_learning --controller joint_mpc --episodes 10
0
python -m eval.report runs/<run_dir>
```

Known Limitations

1. **Async MPC** - Background thread solver is experimental
2. **Task Spec** - Complex guard expressions may have parsing edge cases
3. **Domain Randomization** - Some MuJoCo properties cannot be modified at runtime

---

Next Steps (V13 Roadmap)

- [ ] Neural network policy distillation from contextual bandits
- [ ] Real robot deployment with ROS2 integration
- [ ] Multi-task learning with shared representations
- [ ] Vision-based object detection integration

Contributors

V12 Architecture Refactor - January 2026