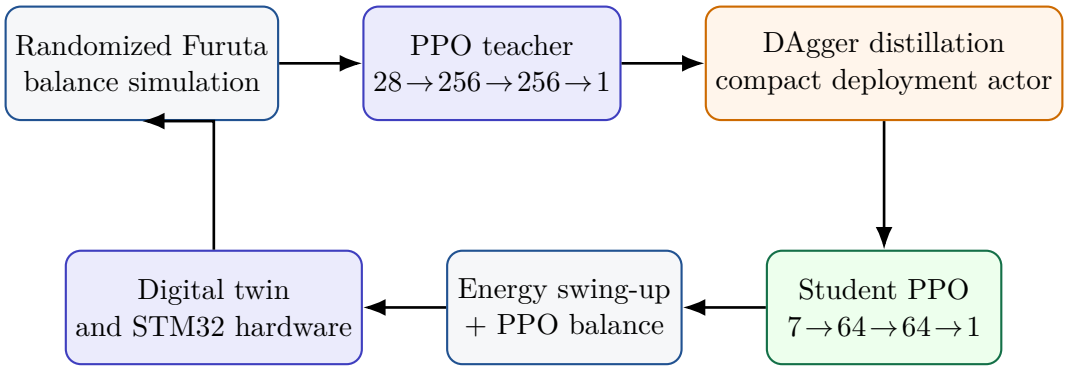
Proximal Policy Optimization Control of a Rotary Inverted Pendulum

Class 12 Laboratory Technical Manual

AI Enabled Control Engineering



Laboratory design objective. Train and evaluate a PPO teacher for robust upright balance, distill the teacher into a compact $7 \to 64 \to 64 \to 1$ actor, verify that actor in the supplied nonlinear digital twin, and deploy it on the rotary inverted pendulum. The physical deployment is a hybrid controller: an energy-based law performs large-angle swing-up, while PPO controls the near-upright region through hysteresis, blending and a discrete Luenberger observer.

Course staff: Professor Xun Huang; TAs: Zhixiang Ju and Haozhe Wang
Version: July 2026

Contents

1 Learning Objectives and Complete Laboratory Architecture

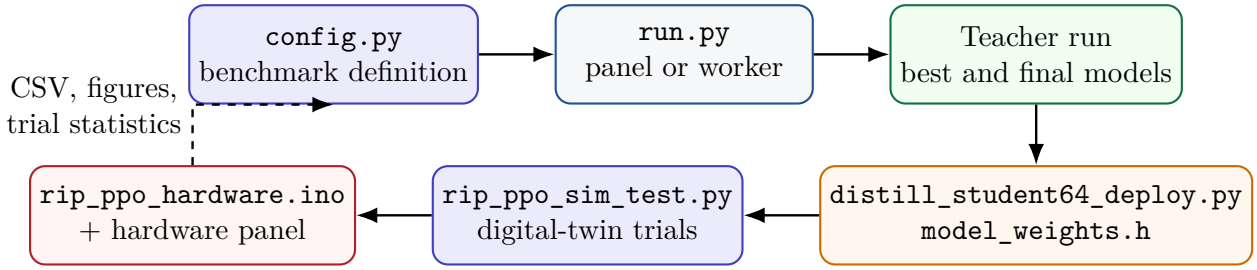
By the end of this laboratory, students should be able to:

1. formulate the rotary inverted pendulum as a Markov decision process;
2. derive the policy-gradient objective and explain why an advantage estimate reduces gradient variance;
3. explain the actor, critic, rollout buffer, generalized advantage estimation and clipped PPO update;
4. interpret the supplied reward function, termination conditions, VecNormalize statistics and domain-randomization curriculum;
5. operate every tab and button in the training panel launched by `pythonrun.py`;
6. distinguish training reward, deterministic evaluation reward, episode completion rate and the randomized best-model score;
7. select a teacher checkpoint and perform DAgger-style distillation into a compact deployment actor;
8. load the resulting `model_weights.h` in the digital-twin panel and execute repeatable simulation trials;
9. flash the supplied STM32 firmware, calibrate sensors, upload the compact actor with CRC verification, and perform controlled hardware trials;
10. preserve configurations, logs and trial-level evidence so that results can be reproduced and compared fairly.

IMPORTANT

The supplied PPO settings are a **benchmark**, not an instruction to change many parameters simultaneously. Modify one controlled factor at a time, preserve the configuration snapshot and logs from every run, and compare repeated simulation and hardware trials. Teaching assistants are not responsible for choosing hyperparameters on behalf of students or explaining every source-code line individually.

1.1 End-to-end artifact flow



2 Reinforcement Learning for Feedback Control

2.1 Agent, environment, observation, action and reward

At policy step t, the environment supplies an observation o_t. The agent samples or selects a continuous action a_t, the simulator advances the physical state, and a scalar reward r_t evaluates the transition. For the benchmark, the action is normalized to $[-1, 1]$ and mapped to motor PWM.

The discounted return is

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}, \qquad 0 < \gamma < 1. \tag{1}$$

The learning objective is the expected return under policy π_θ:

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1}\gamma^t r_t\right]. \tag{2}$$

A trajectory τ contains the observations, actions, rewards and terminal flags produced by interacting with the environment.

2.2 Markov decision process and partial observability

A Markov decision process is described by $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$. The Markov property requires that the current state contain all information needed to predict the next-state distribution. Real actuator lag, command history and uncertain dynamics can violate that assumption when only one angle sample is observed. The teacher therefore uses a history observation:

$$4 \times [\sin\theta, \cos\theta, \dot{\theta}, \sin\alpha, \cos\alpha, \dot{\alpha}] \quad + \quad 4 \text{ previous actions}, \tag{3}$$

for a total of 28 inputs. The compact deployment student instead uses seven current quantities, including the previous applied action, and learns to imitate the teacher over the states visited by both policies.

2.3 Why continuous-action RL is relevant

The motor command is naturally continuous. A policy can express small corrective commands near upright and larger commands when recovering from disturbances. PPO is attractive because it:

- optimizes a stochastic continuous policy directly;
- uses a critic to estimate long-horizon consequences;
- constrains each policy update through a clipped surrogate objective;
- works with parallel environments and batched rollouts;
- integrates naturally with domain randomization and observation/reward normalization.

PPO remains an on-policy method: old data are not stored indefinitely in a replay buffer. A rollout is collected, used for several optimization epochs, and then replaced by data from the updated policy.

3 From Policy Gradient to PPO

3.1 Stochastic policy and likelihood-ratio gradient

Let $\pi_\theta(a \mid o)$ be a policy parameterized by neural-network weights θ. The policy-gradient theorem gives a gradient proportional to

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t \mid o_t) Q^{\pi_\theta}(o_t, a_t)\right]. \tag{4}$$

The identity $\nabla_\theta \pi_\theta = \pi_\theta \nabla_\theta \log \pi_\theta$ allows sampled trajectories to estimate this gradient without differentiating the simulator dynamics.

3.2 Baseline, value function and advantage

Subtracting a state-dependent baseline does not bias the policy gradient. PPO uses a critic $V_\phi(o_t)$ and an advantage estimate

$$A_t = Q(o_t, a_t) - V(o_t). \tag{5}$$

Positive advantage means the sampled action performed better than the critic's expectation; negative advantage means it performed worse. The actor increases the probability of positive-advantage actions and decreases the probability of negative-advantage actions.

3.3 Temporal-difference residual and generalized advantage estimation

The one-step temporal-difference residual is

$$\delta_t = r_t + \gamma V_\phi(o_{t+1}) - V_\phi(o_t). \tag{6}$$

Generalized advantage estimation (GAE) forms

$$\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} = \sum_{l=0}^{T-t-1} (\gamma\lambda)^l \delta_{t+l}. \tag{7}$$

The parameter λ trades variance against bias. The benchmark uses $\gamma = 0.9975$ and $\lambda = 0.95$. Advantages are normalized before the policy update.

3.4 Probability ratio and unconstrained surrogate

A rollout is generated by the old policy $\pi_{\theta_{\mathrm{old}}}$. During optimization, PPO evaluates the probability ratio

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid o_t)}{\pi_{\theta_{\mathrm{old}}}(a_t \mid o_t)}. \tag{8}$$

The ordinary policy-gradient surrogate is

$$L^{\mathrm{PG}}(\theta) = \mathbb{E}_t\left[r_t(\theta)\hat{A}_t\right]. \tag{9}$$

Without a trust-region mechanism, repeated minibatch updates can change the policy too far and destroy a previously useful controller.

3.5 Clipped PPO objective

PPO limits beneficial-looking but excessively large probability changes using

$$L^{\mathrm{CLIP}}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right]. \tag{10}$$

The benchmark uses $\epsilon = 0.2$. The minimum creates a pessimistic objective: when an update would move the probability ratio outside the clipping interval, the clipped term limits the incentive for further movement.

3.6 Critic loss, entropy and complete optimization objective

The critic fits a bootstrapped return target $\hat{R}_t$:

$$L_V(\phi) = \mathbb{E}_t\left[(V_\phi(o_t) - \hat{R}_t)^2\right]. \tag{11}$$

The policy entropy $\mathcal{H}[\pi_\theta(\cdot \mid o_t)]$ encourages exploration. Stable-Baselines3 combines these terms conceptually as

$$L_{\mathrm{total}} = -L^{\mathrm{CLIP}} + c_v L_V - c_e \mathcal{H}, \tag{12}$$

with benchmark coefficients $c_v = 0.5$ and $c_e = 0.002$. Gradient norm is clipped to 0.5. A target KL value of 0.035 provides an additional early-stopping guard inside an update epoch.

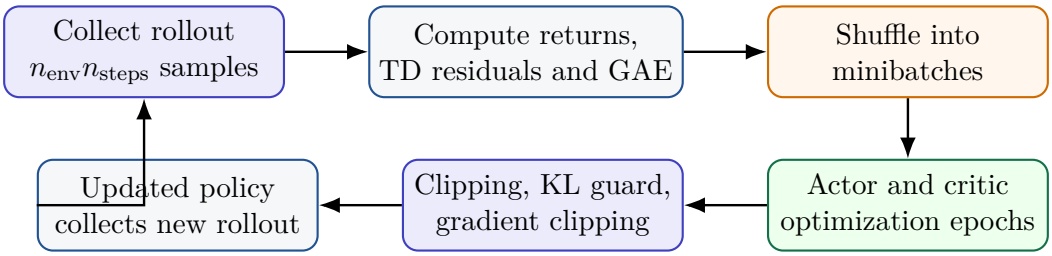
3.7 Actor and critic networks

The teacher has separate multilayer perceptrons for policy and value estimation:

$$\text{Actor: } 28 \to 256 \to 256 \to 1, \qquad \text{Critic: } 28 \to 256 \to 256 \to 1. \tag{13}$$

Both hidden networks use Tanh. The actor represents the mean of a Gaussian policy; the trainable log standard deviation is initialized at -0.7. Deterministic evaluation uses the policy mean rather than a sampled action.

3.8 One complete PPO iteration



For the benchmark, one rollout contains

$$16 \text{ environments} \times 1024 \text{ steps} = 16384 \text{ samples}. \tag{14}$$

Those samples are divided into minibatches of 1024 and reused for 10 epochs. After the update, they are discarded and the new policy collects the next rollout.

3.9 PPO compared with DQN

Property	DQN benchmark	PPO benchmark
Action	Ten discrete PWM choices	Continuous normalized motor command
Policy representation	Q values for every action	Gaussian actor plus value critic
Data reuse	Replay buffer, off-policy	Current rollout only, on-policy
Exploration	ϵ-greedy	Policy standard deviation and entropy
Stabilization	Target/behavior networks	Clipped ratio, GAE, KL guard
Deployment teacher	Small enough to deploy directly in the DQN lab	Large history teacher requires compact distillation
Failure mode to watch	Q-value collapse or replay contamination	Policy regression, poor exploration or loss of rare successful states

4 Benchmark PPO Task and Implementation

4.1 Task scope: upright balance, not learned swing-up

The training package registers `RIPSim2RealBalance-v0`. Each episode begins near the upright equilibrium:

- rotary arm mean 0°, standard deviation 3°;
- pendulum mean 0° from upright, standard deviation 5°;
- nominal initial angular velocities are zero.

The episode terminates if the pendulum exceeds 45° from upright or if a configured state safety limit is violated. Therefore, the PPO teacher is an **upright-region balance controller**. Complete hardware swing-up is provided later by the energy controller in the deployment firmware.

4.2 Timing and action

Quantity	Benchmark value
Physical integration step	0.005 s
Physical rate	200 Hz
Action repeat	1
Policy update rate	200 Hz
Maximum physical steps	3000
Episode duration	15 s
Action type	continuous $[-1, 1]$
Nominal action scale	PWM ± 255

4.3 Teacher observation

The configured observation is `trig_hist4_act4`. A single trigonometric frame is

$$[\sin\theta, \cos\theta, \dot{\theta}, \sin\alpha, \cos\alpha, \dot{\alpha}]. \tag{15}$$

Four frames contribute 24 numbers and four action-history entries contribute four more, giving 28 inputs. Velocity values are clipped by the environment's observation logic. The vectorized environment is wrapped by VecNormalize, which maintains running observation statistics and also normalizes rewards used in PPO optimization.

IMPORTANT

The saved PPO ZIP alone is not the complete teacher. For correct evaluation or distillation, use the matching `vecnormalize.pkl`. The best checkpoint stores this file inside `best_model/`; the final checkpoint stores it in the run root.

4.4 Exact benchmark reward

Angles are wrapped to $[-\pi, \pi]$, angular velocities are clipped to their configured limits, and the actual PWM is normalized. The per-step raw reward is

$$r_t = 0.25 + 8\cos\alpha_t - 2\alpha_t^2 - 0.015\dot{\alpha}_t^2 \tag{16}$$

$$- 0.15\theta_t^2 - 0.01\dot{\theta}_t^2 - 0.0005u_t^2 - 3\,\mathbb{I}(|\alpha_t| > 20^\circ), \tag{17}$$

where u_t is actual PWM divided by the current continuous PWM limit. Completely upright, motionless and centered gives approximately 8.25 per step. The reward is deliberately a benchmark and may be studied through controlled ablations.

The panel plots raw episode reward per step from Monitor information. PPO itself can still train with normalized rewards because VecNormalize is outside the Monitor wrapper. Deterministic evaluation disables reward normalization and reports raw reward per step.

4.5 Termination and displayed success rate

Configured safety limits are:

$$|\theta| \le 12\pi, \qquad |\dot{\theta}| \le 45, \qquad |\dot{\alpha}| \le 40, \tag{18}$$

with termination at $|\alpha| > 45^\circ$. In the panel, one balance evaluation is counted as successful if it completes all 3000 policy steps without early termination. Thus

$$\text{success rate} = 1 - \text{terminated rate}. \tag{19}$$

This is an episode-completion metric, not a separate angle-RMS specification. Always inspect $|\alpha|$, $|\theta|$, angular velocities and PWM together with success rate.

4.6 Domain-randomization curriculum

The training randomization level follows

$$\ell(k) = \ell_0 + \text{clip}\left(\frac{k}{fN}, 0, 1\right)(\ell_f - \ell_0), \tag{20}$$

with $\ell_0 = 0$, $\ell_f = 0.8$, curriculum fraction $f = 0.2$ and total steps $N = 2{,}000{,}000$. Therefore, the level ramps from 0 to 0.8 during the first 400,000 timesteps and remains at 0.8 afterward.

Randomized categories include:

- gravity, masses, lengths, inertias, friction and motor parameters;
- initial-state spreads;
- PWM limit, gain, bias, deadzone, noise and actuator lag;
- angle biases, sensor noise, velocity noise and encoder quantization;
- probabilistic use of a low-pass velocity estimate during training.

The fixed evaluation pair always includes a nominal environment and a second environment at randomization level 0.75.

4.7 Benchmark PPO hyperparameters

Parameter	Value	Interpretation
Total timesteps	2,000,000	Aggregate vectorized environment steps
Parallel environments	16	DummyVecEnv by default
Rollout length	1024 per environment	16,384 samples per rollout
Batch size	1024	16 minibatches per epoch
Optimization epochs	10	Reuse each rollout for 10 passes
Learning rate	3×10^{-4}	Adam optimizer rate
Discount γ	0.9975	Long-horizon return weighting
GAE λ	0.95	Advantage bias-variance tradeoff
Clip range	0.2	PPO probability-ratio clipping
Target KL	0.035	Stops an overly large update
Entropy coefficient	0.002	Exploration regularization
Value coefficient	0.5	Critic-loss weight
Max gradient norm	0.5	Gradient clipping
Actor/Critic hidden sizes	256, 256	Separate Tanh MLPs
Initial log standard deviation	-0.7	Initial Gaussian action spread
Observation normalization	enabled	VecNormalize running statistics
Reward normalization	enabled	Training only; evaluation uses raw reward

4.8 Evaluation score and best-model rule

Every evaluation computes mean reward, mean length, reward per step, episode completion, mean absolute states and normalized PWM. The model-selection score is

$$S = \overline{R} + 0.02\overline{L} - 40\overline{|\alpha|} - 0.4\overline{|\dot{\alpha}|} \tag{21}$$

$$- 0.1\overline{|\theta|} - 5\,p_{\text{terminated}}. \tag{22}$$

The checkpoint in `best_model/` is updated only when the **randomized** evaluation score improves. The final model is merely the policy at the end of training and can be worse than the stored best model.

5 Training Package: File-by-File Reference

5.1 Package structure

```
ppo/
  run.py                          # only user-facing training entry
  config.py                       # all benchmark settings
  config_editor.py                # safe config-file update helper
  training_panel.py               # Tkinter panel imported by run.py
  distill_student64_deploy.py     # teacher-to-student deployment conversion
  rip_env/                        # Furuta simulation environment
  stable_baselines3/              # bundled local SB3 source
  third_party/                    # bundled local Python dependencies
  runs/                           # generated experiments
```

5.2 What each Python file does

File	Responsibility and how it is used
`run.py`	Unique training entry. With no arguments it opens the panel. With `train`, `smoke`, or `eval` it runs the corresponding worker directly. It creates environments, PPO, callbacks, checkpoints and evaluation logs.
`config.py`	Single source of truth for simulator timing, task initialization, reward, PPO, normalization, randomization, evaluation, smoke settings and panel settings.
`config_editor.py`	Used by the panel to update selected values in `config.py` and preserve a backup. Users normally do not run it directly.
`training_panel.py`	Tkinter interface for editing settings, launching the same Python interpreter, displaying logs, progress, reward curves and success rates. Users launch it indirectly through `run.py`.
`distill_student64_deploy.py`	Opens a teacher-run folder chooser, loads PPO plus VecNormalize, performs DAgger-style distillation, evaluates the compact student and writes `model_weights.h`.
`rip_env/`	Contains the nonlinear environment, observation history, dynamics, randomization and logging implementation. Treat it as experiment infrastructure unless an assignment explicitly asks for environment modification.
`stable_baselines3/`	Bundled SB3 implementation. `run.py` places this local source ahead of a global SB3 installation. Do not edit it for normal laboratory tuning.
`third_party/`	Bundled supporting dependencies. Keep this directory with the project.

5.3 Python environment and dependencies

Use the Python interpreter that already contains PyTorch, NumPy, Matplotlib and Tk support. The project supplies local SB3 and supporting packages. A typical launch is:

```
cd /path/to/ppo
/path/to/python run.py
```

If the terminal command `python` is unavailable, use `python3` or the absolute interpreter path from the course virtual environment. Do not remove `stable_baselines3/`, `third_party/`, or `rip_env/`.

6 Operating the PPO Training Panel

6.1 Opening the panel

The only normal entry is:

```
python run.py
```

The process imports `TrainingPanel` and opens the GUI. The toolbar contains:

Button	Operation
Save to `config.py`	Parses visible fields, writes them into `config.py`, creates a backup, and reloads the configuration.
Reload	Discards unsaved GUI text and reloads values from the current file.
Start Full Training	Auto-saves visible settings, launches `run.pytrain` using the same Python interpreter, and switches to the live-curves tab.
Smoke Test	Launches a shortened end-to-end training run using the smoke overrides. It verifies the pipeline, not final performance.
Stop	Terminates the current child training or evaluation process.
Open `runs`	Opens the experiment-output directory in the operating system.

6.2 Tab 1: Common Parameters

This tab exposes frequently changed values grouped into environment, sim-to-real randomization, PPO, reward and evaluation. Each row shows a label, editable value and exact dotted config path. Examples include `PPO.learning_rate`, `REWARD.k_cos`, and `DOMAIN_RANDOMIZATION.param_scale_ranges.m2_scale`.

Tuples must be entered using Python syntax, for example `(256,256)` or `(0.9,1.1)`. Boolean values use `true/false`. The panel writes these values into `config.py`; it does not maintain a separate hidden settings file.

6.3 Tab 2: All Config Parameters

The advanced tree flattens all supported configuration dictionaries. Select a path, edit its value, and apply it. Use this tab for less common fields that do not appear in the common tab. Reload after an external edit to avoid overwriting new values with stale GUI text.

6.4 Tab 3: Training, Evaluation and Curves

The top status block displays the current run directory, latest completed training episode and latest deterministic evaluation. The three plots are:

1. **PPO Episode Reward / Step**: individual raw episode reward per step and a 30-episode moving mean;
2. **Deterministic Evaluation Reward / Step**: nominal and level-0.75 randomized evaluations;
3. **Evaluation Success Rate**: fraction of evaluation episodes that completed the full balance horizon.

The training line also shows a rolling 30-episode completion rate. Because training uses stochastic actions and evaluation uses deterministic actions, training reward can improve while deterministic evaluation degrades. Model selection should prioritize deterministic randomized metrics rather than the training curve alone.

6.5 Tab 4: Full Log

This tab contains all stdout and error output from the worker. Important machine-readable lines include:

```
[RUN_DIR] path=...
[PROGRESS] algorithm=PPO timesteps=... total=... percent=... fps=... eta_s=...
[TRAIN_EPISODE] timesteps=... reward_per_step=... moving_mean_30=...
[EVAL_METRICS] timesteps=... eval_type=nominal ...
[EVAL_METRICS] timesteps=... eval_type=randomized ...
[BEST] saved ... randomized_score=...
```

Use this tab when a curve appears frozen, a dependency fails, or training exits unexpectedly.

6.6 Tab 5: Model Evaluation

Choose an SB3 model ZIP and click Run Evaluation. The panel launches `run.pyeval<model.zip>`. The loader looks for `vecnormalize.pkl` in the same directory as the model. Therefore select `best_model/best_model.zip` together with its sibling normalization file, or `final_model.zip` in the run root.

6.7 Recommended panel workflow

1. Copy the original package to a new experiment directory.
2. Run the smoke test before any long training run.
3. Save the benchmark configuration and take a screenshot or copy the generated `config_snapshot.json`.
4. Start full training and confirm that FPS, completed episodes and deterministic evaluations appear.
5. Inspect nominal and randomized reward, success and angle error together.
6. At the end, compare `best_model/best_model.zip` with `final_model.zip`; do not assume the final policy is best.
7. Evaluate the chosen teacher explicitly before distillation.

6.8 Command-line modes

The GUI is the normal interface, but equivalent commands remain available:

```
# Open panel
python run.py

# Full benchmark training
python run.py train

# Short integration test
python run.py smoke

# Deterministic randomized evaluation
python run.py eval runs/<run>/best_model/best_model.zip
```

7 Training Outputs and How to Interpret Them

7.1 Run-directory layout

A typical run contains:

```
runs/ppo_sb3_sim2real_balance_YYYYMMDD_HHMMSS/
  config_snapshot.json
  env_config_snapshot.json
  training_progress.json
  training_metrics.csv
  monitor_train/
```

```
env_logs/
models/
  ppo_stage1_balance_<steps>_steps.zip
  ppo_stage1_balance_vecnormalize_<steps>_steps.pkl
eval_logs/
  eval_metrics.csv
  eval_curve.png
best_model/
  best_model.zip
  vecnormalize.pkl
final_model.zip
vecnormalize.pkl
```

7.2 Important files

File	Interpretation
`config_snapshot.json`	Exact user-level configuration captured before training.
`env_config_snapshot.json`	Expanded environment object after configuration construction.
`training_metrics.csv`	Per-completed-episode reward, length, reward per step, moving mean and rolling completion rate.
`training_progress.json`	Most recent timesteps, percentage, FPS, elapsed time and ETA.
`monitor_train/`	SB3 Monitor CSVs from each parallel environment.
`models/`	Periodic policy and VecNormalize checkpoints. Useful for diagnosing policy regression.
`eval_logs/eval_metrics.csv`	Nominal and randomized evaluation metrics at each evaluation point.
`eval_logs/eval_curve.png`	Nominal and randomized score history.
`best_model/best_model.zip`	Highest randomized evaluation score observed so far.
`best_model/vecnormalize.pkl`	Matching normalization statistics for the best model.
`final_model.zip`	Last policy after the final PPO update; not necessarily best.
`vecnormalize.pkl`	Normalization statistics matching the final model.

7.3 Reading a healthy training result

A useful balance teacher should show:

- a rising and then stable deterministic randomized reward per step;
- high full-episode completion rate in both nominal and randomized tests;
- small mean $|\alpha|$ and controlled $|\theta|$;
- no persistent collapse after the randomization level reaches 0.8;
- a best-model checkpoint that is not merely an early accidental spike.

Run more than one random seed before concluding that a parameter change is better.

8 Teacher-to-Student Distillation

8.1 Why distillation is required

The trained teacher uses 28 inputs, two 256-unit hidden layers and VecNormalize. The deployment firmware accepts a compact float32 actor with seven inputs and two 64-unit hidden layers. Distillation transfers the teacher action mapping to the smaller actor while embedding compact-observation mean and standard deviation inside the generated header.

8.2 Compact deployment observation

The student input is

$$[\sin\theta, \cos\theta, \dot{\theta}, \sin\alpha, \cos\alpha, \dot{\alpha}, u_{t-1}], \tag{23}$$

where u_{t-1} is the previous applied controller PWM divided by the model PWM scale and clipped to $[-1, 1]$. The generated header normalizes each component:

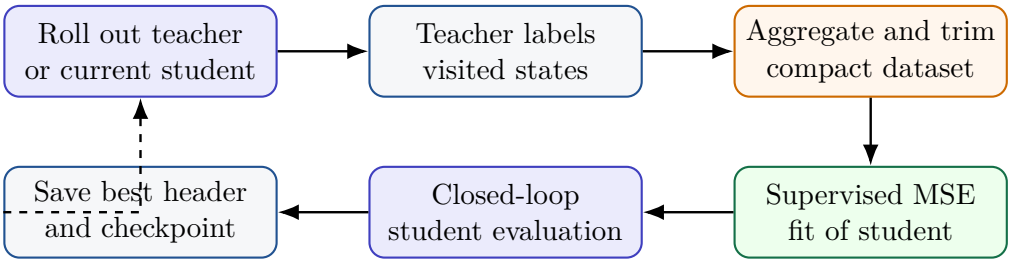
$$z_i = \text{clip}\left(\frac{o_i - \mu_i}{\sigma_i}, -10, 10\right). \tag{24}$$

The deployment actor is

$$7 \to 64 \xrightarrow{\tanh} 64 \xrightarrow{\tanh} 1 \xrightarrow{\tanh} [-1, 1]. \tag{25}$$

8.3 DAgger-style data collection

The first iteration can roll out the teacher. Later iterations roll out the current student but label every visited compact observation with the teacher's deterministic action. This reduces distribution shift: the student is trained not only on states the teacher naturally visits, but also on states caused by student errors.



The supervised objective is

$$L_{\text{distill}} = \frac{1}{N}\sum_{i=1}^{N} \|\pi_{\text{student}}(o_i^{(7)}) - \pi_{\text{teacher}}(o_i^{(28)})\|_2^2. \tag{26}$$

8.4 Default distillation settings

Setting	Default	Meaning
DAgger iterations	8	Repeated collect-fit-evaluate cycles
Steps per iteration	40,000	New state-action labels per cycle
Maximum dataset	320,000	Random trim after aggregation
Collection levels	0, 0.25, 0.50, 0.75	Robustness coverage
Epochs per iteration	25	Supervised passes
Batch size	4096	Student-fitting minibatch
Learning rate	8×10^{-4}	AdamW rate
Validation fraction	0.10	Held-out imitation metrics
Closed-loop evaluation	64 episodes	Student, not teacher, drives the environment
Evaluation level	0.75	Robust deployment emphasis

8.5 Using the distillation script

Normal graphical use:

```
cd /path/to/ppo
python distill_student64_deploy.py
```

A native folder chooser appears. Select the teacher **run directory**, not a generic backup ZIP. The script searches in this order:

1. `best_model/best_model.zip`;
2. `selected_best_model.zip`;
3. `best_model.zip`;
4. `final_model.zip`;
5. another valid SB3 PPO archive found below the directory.

It also searches for the matching VecNormalize file.

Command-line alternatives are:

```
python distill_student64_deploy.py \
  --teacher-dir /path/to/teacher_run --no-gui

python distill_student64_deploy.py \
  --teacher /path/to/best_model.zip --no-gui

# Integration test only; do not deploy its output
python distill_student64_deploy.py \
  --teacher-dir /path/to/teacher_run --no-gui --smoke
```

8.6 Distillation outputs

The selected teacher run receives:

```
<teacher_run>/
  model_weights.h                  # primary deployment file
  distill_student64_deploy/
    distill_config_snapshot.json
    distill_log.csv
    student64_deploy.pt
    student64_deploy_weights.npz
    best/
      model_weights.h
      student64_deploy.pt
      student64_deploy_weights.npz
```

The main deployment artifact is `model_weights.h`. The `--smoke` option validates loading, collection, fitting and export, but its tiny dataset is not a valid final controller.

9 Deployment Package and Hybrid Controller

9.1 Package files

```
deploy_ppo/
  rip_ppo_sim_test.py
  rip_ppo_hardware.py
  rip_ppo_hardware/
    rip_ppo_hardware.ino
```

File	Purpose
`rip_ppo_sim_test.py`	Loads the compact header or cached NPZ, simulates the nonlinear Furuta plant, performs energy swing-up and PPO hand-off, shows a live 3-D view, writes CSV and result figures, and supports headless repeated trials.
`rip_ppo_hardware.py`	PyQt5 serial panel for port selection, model upload, CRC verification, calibration, controller settings, GO/STOP, live 3-D display and hardware logging.
`rip_ppo_hardware.ino`	STM32 firmware containing the 200 Hz control timer, sensor conversion, energy swing law, Luenberger observer, compact PPO inference, safety checks and protocol-v11 model upload.

9.2 Why the deployed controller is hybrid

The teacher PPO was trained near upright. The deployment therefore uses:

$$\text{natural hanging state} \rightarrow \text{energy swing-up} \rightarrow \text{blended hand-off} \rightarrow \text{PPO upright balance}. \tag{27}$$

Far from upright, the firmware applies an initial kick and then a phase-based pumping command. Near upright, it activates the PPO actor. If the pendulum leaves the exit angle, the system returns to swing-up.

9.3 Energy swing-up law

During the initial kick interval,

$$u_{\text{swing}} = +u_s. \tag{28}$$

Afterward, the firmware evaluates

$$q = \dot{\alpha}_{\text{LPF}} \cos \alpha \tag{29}$$

and applies a signed swing PWM according to the phase. The default magnitude is 120 with a safety limit of 150. This law is inherited from the MPC-firmware large-angle strategy; it is not learned by PPO.

9.4 Hysteresis and smooth blending

Default switching angles are:

$$\text{enter PPO if } |\alpha| \le 15^\circ, \qquad \text{exit PPO if } |\alpha| \ge 25^\circ. \tag{30}$$

The gap prevents rapid mode chatter. A blend state obeys

$$b_{k+1} = \text{clip}\left[b_k + \lambda(b_{\text{target}} - b_k), 0, 1\right] \tag{31}$$

with default $\lambda = 0.18$. The applied command is

$$u_k = (1 - b_k)u_{\text{swing},k} + b_k u_{\text{PPO},k}, \tag{32}$$

then clipped to the safety PWM limit.

9.5 Near-upright Luenberger observer

When PPO activates, the firmware initializes a discrete observer. Its update has the form

$$\hat{x}_{k+1} = (A_d - LC)\hat{x}_k + B_d u_k + L y_k, \tag{33}$$

where $y_k = [\theta_k, \alpha_k]^T$. The observer supplies $[\theta, \dot{\theta}, \alpha, \dot{\alpha}]$ to the compact actor. If the observer becomes non-finite or its angle estimate disagrees excessively with measurement, it is reinitialized. Far from upright, finite-difference velocities are filtered by the panel-configured large-angle LPF coefficient.

9.6 Compact model inference

The float32 model uploaded to STM32 contains:

- $W_0, b_0, W_1, b_1, W_2, b_2$;
- observation means and inverse standard deviations;
- observation clip value and model PWM scale.

The blob is 19,012 bytes. Firmware inference performs normalization, Tanh hidden layers and final Tanh output. The resulting normalized action is multiplied by the model PWM scale before blending and final safety clipping.

10 Using the Digital-Twin Simulation Panel

10.1 Dependencies and launch

The panel uses NumPy, PyQt5 and Matplotlib. Launch it from the deployment folder:

```
cd /path/to/deploy_ppo
python rip_ppo_sim_test.py
```

10.2 Model selection

The panel accepts:

- the generated `model_weights.h`;
- a compatible cached `.ppo64_deploy.npz`;
- a run or deploy directory containing one compatible header.

When a header is loaded, the script validates dimensions $7 \to 64 \to 64 \to 1$, parses weights and normalization, and may create a cached NPZ next to the header.

10.3 Simulation settings

Control	Default	Meaning
Initial condition	Random hanging-down	Exact down, random down, or near-upright options
Duration	30 s	Simulation horizon
Playback speed	1.0	Wall-clock visualization speed; dynamics remain 5 ms
Randomization level	0.0	Samples physical parameters from the built-in ranges
Seed	2026	Reproducible initialization and randomization
Safety PWM max	150	Final command clipping
Energy swing PWM	120	Large-angle swing magnitude
Initial kick	0.10 s	Initial positive command duration
PPO enter angle	15°	Activates observer and PPO
PPO exit angle	25°	Returns to energy swing
Blend coefficient	0.18	Hand-off speed
Velocity LPF	0.25	Large-angle finite-difference velocity filter
Generate CSV log	enabled	Writes all state, mode and policy fields

10.4 Button sequence

1. Click **Choose Model File** or **Choose Run Folder**.

2. Check the resolved architecture, model PWM scale and model ID.
3. Set initial condition, duration, randomization, controller limits and output directory.
4. Click **SAVE / Load Model & Apply Settings**. Any later setting change marks the configuration dirty and requires SAVE again.
5. Click **GO**. The animation runs the 200 Hz dynamics while the right panel shows angles, PWM, mode, blend, action and raw policy output.
6. Click **STOP** to terminate early or **RESET** to return to the selected initial condition.
7. Use **Show Last Result Curves** after a completed or stopped run.

The result figure contains pendulum angle, rotary-arm angle, applied PWM and controller mode/blend. The CSV fields are time, state, PWM, mode, blend, normalized action and raw policy output.

10.5 Headless simulation

For scripted repeated trials:

```
python rip_ppo_sim_test.py --headless \
  --model /path/to/model_weights.h \
  --duration 30 \
  --seed 2026 \
  --randomization 0.0 \
  --pwm-limit 150 \
  --swing-pwm 120
```

The terminal prints the number of steps, final pendulum angle, final stable-start time and capture count. Change only the seed for a nominal ten-trial batch; use a documented randomization level for robustness trials.

10.6 Recommended simulation acceptance procedure

Perform at least ten trials with the exact model intended for hardware. Record:

- whether the pendulum entered the PPO region;
- time of final capture and stable duration;
- number of recaptures;
- mean and standard deviation of $|\alpha|$ in the final stable phase;
- mean and standard deviation of |PWM|;
- maximum $|\theta|$ and whether any safety termination occurred.

A single successful animation is not sufficient evidence.

11 Preparing the STM32 Firmware

11.1 Flash before opening the hardware panel

Open the folder `rip_ppo_hardware/` in Arduino IDE and compile `rip_ppo_hardware.ino` for the course STM32 target. Upload the firmware, then close Arduino Serial Monitor before opening the Python panel. Both programs cannot safely own the same serial port simultaneously.

The bundled panel expects protocol version 11 and a startup line equivalent to:

```
READY,RIP_PPO64_HYBRID_HW_STABLE,11
```

A version mismatch means the Python panel and firmware are not from the same package.

11.2 Firmware responsibilities

The firmware executes:

- a hardware-timer control tick at 200 Hz;
- encoder and potentiometer acquisition;
- calibration-based conversion of the pendulum angle;
- large-angle velocity LPF and near-upright Luenberger estimation;
- energy swing, PPO switching and command blending;
- compact neural-network inference;
- motor direction, PWM clipping, braking and standby control;
- safety termination;
- model upload, CRC checking, configuration, telemetry and calibration commands.

11.3 Model upload protocol

The hardware panel uses baud 921600 and sends the float32 blob as ASCII-hex chunks. The default chunk is 96 bytes. Each chunk carries offset, length and CRC32. STM32 acknowledges a chunk only after decoding, validating and committing it. After all chunks, the host and device compare the full-model CRC and model ID. The GO button remains disabled until both model upload and configuration are acknowledged.

12 Using the Physical-System Panel

12.1 Launch

Install PyQt5, pyserial, NumPy and Matplotlib in the deployment Python environment, then run:

```
cd /path/to/deploy_ppo
python rip_ppo_hardware.py
```

12.2 Serial Connection group

Click **Refresh**, select the STM32 serial device and click **Connect**. Wait until the panel reports that stable protocol v11 is ready. If no port appears, check the USB cable, board power, driver, firmware and whether another program has the port open.

12.3 PPO Model Selection and STM32 Deployment group

Choose either `model_weights.h`, a compatible cached NPZ, or a run/deploy folder. The preview should report:

$$\text{compact7 } 7 \to 64 \to 64 \to 1 \text{ Tanh}, \tag{34}$$

internal normalization, a model ID, and a 19,012-byte float32 blob. Do not use the original 28-input teacher ZIP in this panel.

12.4 Controller settings

Setting	Default	Operational meaning
Duration	30 s	Firmware stops automatically at the horizon
Safety PWM max	150	Final absolute motor limit
Energy swing PWM	120	Large-angle phase-pumping magnitude
Initial kick	0.10 s	Initial positive swing command
PPO enter	15°	Activate observer and PPO
PPO exit	25°	Deactivate PPO and return to swing
Blend λ	0.18	Smooth transition coefficient
Large-angle velocity LPF β	0.25	Angle-difference velocity filtering outside PPO region

Changing any field marks the setup dirty. SAVE is required again before GO.

12.5 Sensor calibration

The panel shows current potentiometer and encoder raw values.

1. Hold the pendulum exactly upright and click **Hold upright** → **Calibrate** $\alpha = 0$.
2. Let the pendulum hang naturally downward and click **Hang down** → **Calibrate** $\alpha = \pi$.
3. Verify that the two raw references differ substantially.
4. Confirm the encoder radians-per-count value and motor sign for the physical device.

Calibration is part of the uploaded configuration. Recalibration therefore requires SAVE before the next GO.

12.6 SAVE: model and parameter upload

Click **SAVE / Upload PPO Model & Parameters**. The panel:

1. parses and validates the model;
2. builds the float32 blob;
3. sends a begin command with byte count, model ID and CRC;
4. sends CRC-checked 96-byte chunks;
5. verifies the final device CRC and model ID;
6. sends controller duration, PWM, switching, calibration and motor-sign configuration;
7. enables GO only after both acknowledgments succeed.

Do not move the mechanism or press GO while upload is incomplete.

12.7 GO, live display and STOP

Click GO only when the mechanism is clear and an operator is ready to stop it. The right panel displays the real-time mechanism, angles, PWM, controller mode, observer blend, normalized action and raw policy output. Mode values are:

$$0 = \text{disabled}, \quad 1 = \text{energy swing}, \quad 2 = \text{blend}, \quad 3 = \text{PPO balance}. \tag{35}$$

STOP is sent repeatedly to improve the probability of an immediate stop over a busy serial link. The configured duration also ends the run automatically.

12.8 Result logging

The panel writes a PNG and optionally a CSV to the selected output directory. The result figure includes pendulum angle, rotary-arm angle, PWM and a stable-stage PWM distribution. Use **Show Last Result Curves** to reopen the most recent result. Preserve aborted and safety-stopped trials; excluding them would bias the reported success rate.

13 Safety Limits and Error Messages

DANGER

The rotary inverted pendulum can move rapidly. Keep hands, cables and loose objects outside the swept volume. Begin with the supplied PWM limits, keep the STOP control accessible, and use short experiments before longer runs. Never disable a safety termination merely to obtain a higher apparent success rate.

The firmware stops and reports `ERR,SAFETY_STATE` if a measured or filtered state is non-finite, the unwrapped arm exceeds 12π, or a filtered angular velocity exceeds twice its configured training limit. Common causes are wrong encoder scale/sign, poor potentiometer calibration, discontinuous angle conversion, electrical noise or an actual unstable trajectory.

Message or symptom	Meaning and response
Firmware mismatch	Flash the `.ino` bundled with the same hardware panel.
No serial port	Close Serial Monitor, reconnect USB, verify cable and board power, then Refresh.
`ERR,MODEL_REQUIRED`	SAVE has not completed successfully.
`ERR,CONFIG_REQUIRED`	Model may be loaded but controller settings were not acknowledged. SAVE again.
Model chunk timeout	Keep the board powered, close competing serial applications, reconnect and repeat SAVE.
CRC mismatch	Do not run. Repeat upload and inspect serial stability.
`ERR,CAL_WHILE_RUNNING`	Stop the controller before calibration.
`ERR,SAFETY_STATE`	Inspect calibration, scale, sign and the last telemetry values.
`ERR,MODEL_INFERENCE`	Model contains invalid values or inference became non-finite. Re-distill and validate the header.
GO disabled	Connection, model ACK, config ACK or clean-settings condition is missing.
Early automatic stop	Check configured duration and safety status; a 30 s value ends at 30 s by design.

14 Complete Recommended Workflow

1. **Smoke test the trainer.** Open `run.py`, click Smoke Test, and confirm checkpoint, evaluation and curve generation.
2. **Train the teacher.** Preserve the benchmark first; then perform one controlled parameter change per experiment.
3. **Select the teacher.** Inspect randomized deterministic metrics and choose the best checkpoint rather than assuming the final model is best.

4. **Evaluate the teacher.** Use the training panel's Model Evaluation tab with the matching VecNormalize file.
5. **Distill.** Run `distill_student64_deploy.py`, select the teacher run, and wait for the full DAgger process.
6. **Check distillation.** Inspect validation error, closed-loop score, termination rate and the generated `model_weights.h`.
7. **Run ten digital-twin trials.** Use documented seeds and controller settings. Reject any model with unacceptable safety or capture behavior.
8. **Flash firmware.** Use the `.ino` from the same deploy package.
9. **Calibrate and SAVE.** Verify model architecture, CRC, upright and hanging references, encoder scale and motor sign.
10. **Run short hardware trials.** Begin with 5–10 s and conservative PWM. Increase duration only after repeated safe behavior.
11. **Run the required ten physical trials.** Record every trial, including failures, manual stops and safety terminations.

15 Required Experimental Work

15.1 Training-stage requirement

Students should conduct a controlled search rather than one unrecorded long run. For every experiment, preserve:

- complete parameter values and reward definition;
- seed and software/package version;
- training and deterministic evaluation curves;
- best and final checkpoint identities;
- training speed and any interruption;
- an explanation of why the next experiment was chosen.

A useful improvement can be a more stable reward curve, a higher randomized completion rate, smaller angle error, lower control effort, reduced seed sensitivity or better retention after strong randomization.

15.2 Ten simulation trials

Use the same compact `model_weights.h`, fixed controller parameters and a predeclared seed list. Report individual and aggregate results. Do not discard failed runs. Recommended summary statistics include success rate, final capture time, stable duration, mean $|\alpha|$, mean |PWM|, maximum arm angle and safety-stop count.

15.3 Ten physical-system trials

Deploy the student model selected from simulation evidence. Keep calibration, PWM limits, switching angles and test duration fixed across the ten main trials. A model is not required to outperform the instructor model physically, but the report must explain differences between simulation and hardware and must include all unsuccessful trials.

STUDENT TASK

Read the source code and this manual carefully before asking for assistance. Course staff will help diagnose reproducible faults, but will not choose a final reward, PPO setting or model checkpoint on behalf of a student.

16 Troubleshooting Guide

Symptom	Likely cause	Action
`pythonrun.py` does not open panel	Wrong interpreter or missing Tk	Use the course interpreter; verify `python-mtkinter`.
Missing Gymnasium/SB3	Local folders deleted or wrong working directory	Run from project root; restore `third_party/` and `stable_baselines3/`.
No live curves	No completed episodes or no evaluation yet	Check Full Log, training FPS and evaluation frequency.
Evaluation never occurs	Worker stopped early or configuration invalid	Inspect log and `training_progress.json`; current code uses threshold rather than modulo matching.
Training reward high, eval poor	Stochastic policy or normalization hides deterministic weakness	Prioritize deterministic nominal/randomized metrics.
Best model absent	No evaluation completed	Confirm run exceeded evaluation interval and did not crash.
Eval model behaves incorrectly	Missing/mismatched VecNormalize	Use the normalization file saved beside the selected model.
Distillation chooser finds no teacher	Wrong folder or generic backup ZIP	Select a real run directory containing an SB3 model archive.
Distillation is very slow	Full 8-iteration DAgger and 64-episode evaluations	Do not use smoke output for deployment; allow the formal run to finish.
Header rejected	Wrong architecture or malformed header	Use the current distillation script and confirm $7 \to 64 \to 64 \to 1$.
Simulation never enters PPO mode	Swing PWM, initial condition or enter threshold inappropriate	Inspect mode trace and angle; verify model and safety settings.
Hardware port disappears	Cable/power/port ownership issue	Close Serial Monitor, reconnect, Refresh, and use the matching protocol firmware.
SAVE completes but GO disabled	Dirty setting, missing model ACK or missing config ACK	Do not edit fields after SAVE; repeat SAVE and inspect status.
Immediate `ERR,SAFETY_STATE`	Calibration, encoder scale/sign, discontinuity or excessive velocity	Stop, recalibrate and verify raw sensor direction manually.
PPO enters then immediately exits	Excess capture speed or narrow hysteresis	Inspect $\dot{\alpha}$, blend and mode; tune only after simulation evidence.
Large steady arm drift	Model mismatch, calibration bias or insufficient arm penalty	Compare teacher/student, sensor bias and motor sign.

17 Command Quick Reference

```
# TRAINING PANEL
cd /path/to/ppo
python run.py

# TRAINING WORKERS
python run.py smoke
python run.py train
python run.py eval runs/<run>/best_model/best_model.zip

# DISTILLATION GUI
python distill_student64_deploy.py

# DISTILLATION WITHOUT GUI
python distill_student64_deploy.py \
  --teacher-dir runs/<teacher_run> --no-gui

# DIGITAL-TWIN GUI
cd /path/to/deploy_ppo
python rip_ppo_sim_test.py

# DIGITAL-TWIN HEADLESS
python rip_ppo_sim_test.py --headless \
  --model /path/to/model_weights.h --duration 30 --seed 2026 \
  --randomization 0.0 --pwm-limit 150 --swing-pwm 120

# HARDWARE PANEL
python rip_ppo_hardware.py
```

18 Summary

The complete PPO laboratory has four distinct technical layers:

1. a history-based, normalized PPO teacher trained for upright balance;
2. deterministic nominal and randomized evaluation with explicit best-model protection;
3. DAgger-style distillation into a compact $7 \to 64 \to 64 \to 1$ actor with internal normalization;
4. hybrid energy swing-up and PPO upright control in digital twin and STM32 hardware.

Reliable conclusions require more than one attractive reward curve. Preserve configuration evidence, use the matching normalization statistics, test the actual compact model, count every failed trial, and treat safety limits as part of the controller rather than as obstacles to performance.