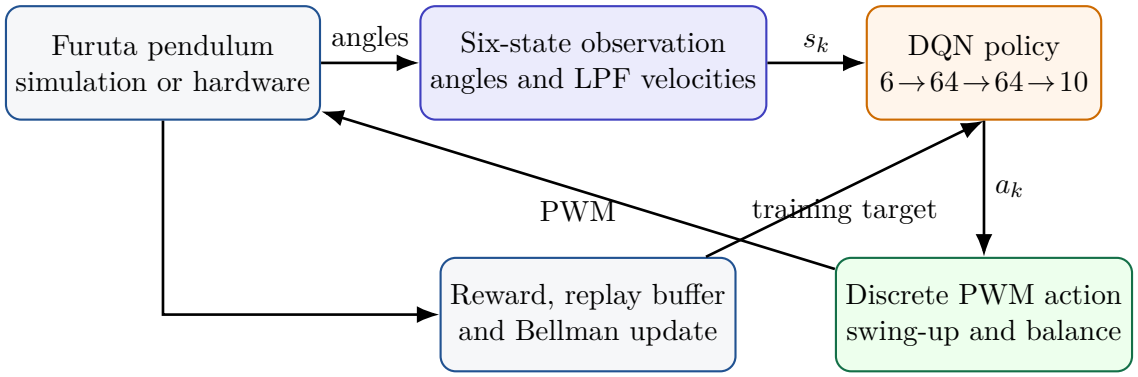
Deep Q-Network Control of a Rotary Inverted Pendulum

Class 11 Laboratory Technical Manual

AI Enabled Control Engineering



Class-11 design objective. Train a single Deep Q-Network to perform the complete rotary-inverted-pendulum task: hanging-down swing-up, upright capture, and long-duration balance. The supplied training package is a reproducible benchmark, not a final answer. Students must improve training stability through controlled hyperparameter experiments, document every run, validate the chosen model in ten simulation trials, and then perform ten physical-system trials with the model they judge to be best.

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

Contents

1 Learning Objectives and Laboratory Architecture

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

1. explain reinforcement learning as sequential decision making from interaction data;
2. formulate the rotary inverted pendulum as a Markov decision process;
3. derive the Bellman optimality equation for the action-value function;
4. explain how Q-learning approximates the Bellman fixed point from samples;
5. explain why DQN uses a neural network, replay buffer, target network, behavior network and ϵ-greedy exploration;
6. read and operate the supplied DQN training panel and deployment panels;
7. tune one group of parameters at a time while preserving all configuration snapshots and training logs;
8. distinguish nominal training performance from robustness under domain randomization;
9. select, test and deploy a trained model without a distillation stage;
10. evaluate performance statistically over repeated simulation and hardware trials rather than from a single successful run.

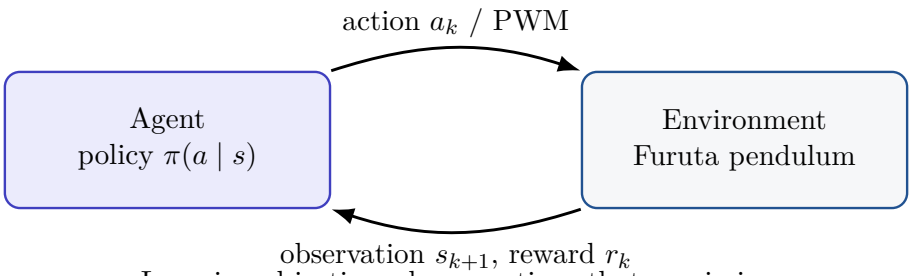
IMPORTANT

The supplied model and parameters are a **benchmark**. The objective is not simply to reproduce the instructor run. Each student should obtain a model whose training curves are more stable, show less policy collapse, or retain performance more successfully after domain randomization begins. A student model does not need to outperform the instructor model on the physical system to satisfy this objective.

2 Reinforcement Learning for Feedback Control

2.1 Agent, environment, state, action and reward

Reinforcement learning (RL) studies how an agent improves a decision policy by interacting with an environment. At time step k, the agent receives a state or observation s_k, chooses an action a_k, receives a scalar reward r_k, and observes the next state s_{k+1}. The objective is not to maximize only the immediate reward; it is to maximize the accumulated future return.



Learning objective: choose actions that maximize the expected discounted return over time.

For a discounted infinite-horizon problem, the return from step k is

$$G_k = \sum_{i=0}^{\infty} \gamma^i r_{k+i}, \qquad 0 \le \gamma < 1, \tag{1}$$

where γ is the discount factor. A larger γ places more weight on long-term consequences.

2.2 Markov decision process

A standard RL problem is described by a Markov decision process (MDP)

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, r, \gamma),$$

where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $p(s' \mid s, a)$ is the transition distribution and $r(s, a, s')$ is the reward. The Markov property states that, given the current state and action, the future transition does not depend on the full earlier history.

For the supplied DQN benchmark, the observation is

$$s_k = \begin{bmatrix} \sin\theta_k & \cos\theta_k & \dot{\theta}_{k,\mathrm{LPF}} & \sin\alpha_k & \cos\alpha_k & \dot{\alpha}_{k,\mathrm{LPF}} \end{bmatrix}^T. \tag{2}$$

The action set is the ten-element PWM set

$$\mathcal{A} = \{-150, -120, -90, -60, -30, 30, 60, 90, 120, 150\}. \tag{3}$$

The same policy is responsible for both swing-up and balance.

2.3 Why RL is useful for the rotary inverted pendulum

The Furuta pendulum is underactuated, nonlinear and open-loop unstable. A swing-up action can temporarily move the system away from the final upright configuration while increasing mechanical energy. RL is useful because it can learn the long-term value of such action sequences directly from interaction data. Unlike a purely local linear controller, the learned policy can represent one nonlinear mapping that covers hanging-down motion, energy pumping, capture and balancing.

> RL does not remove the need for engineering judgement. State construction, reward design, exploration, actuator limits, sampling rate, numerical stability, logging and safe hardware operation remain control-engineering decisions.

3 From Bellman Optimality to DQN

3.1 State-value and action-value functions

For a policy π, the state-value function and action-value function are

$$V^\pi(s) = \mathbb{E}_\pi[G_k \mid s_k = s], \tag{4}$$
$$Q^\pi(s, a) = \mathbb{E}_\pi[G_k \mid s_k = s, a_k = a]. \tag{5}$$

The optimal action-value function is

$$Q^*(s, a) = \max_\pi Q^\pi(s, a). \tag{6}$$

Once Q^* is known, an optimal greedy policy is

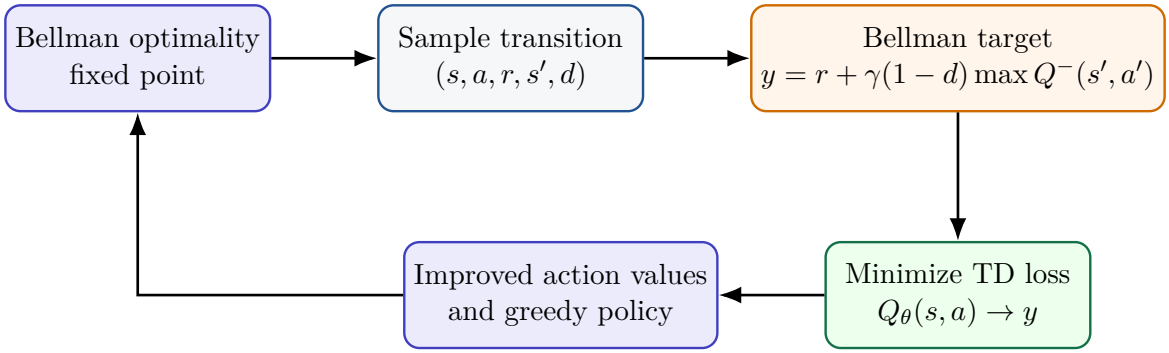
$$\pi^*(s) = \arg\max_{a \in \mathcal{A}} Q^*(s, a). \tag{7}$$

3.2 Bellman optimality equation

Bellman's optimality principle states that an optimal decision must leave the remaining problem in an optimal state. For the optimal action-value function, this gives the fixed-point equation

$$Q^*(s, a) = \mathbb{E}\left[r + \gamma \max_{a' \in \mathcal{A}} Q^*(s', a') \mid s, a \right]. \tag{8}$$

The right-hand side is the immediate reward plus the discounted value of the best action available at the next state.



3.3 Q-learning as sampled Bellman iteration

When the transition model is unavailable, the expectation in (8) is replaced by observed samples. Tabular Q-learning uses

$$Q(s_k, a_k) \leftarrow Q(s_k, a_k) + \eta \left[r_k + \gamma \max_{a'} Q(s_{k+1}, a') - Q(s_k, a_k) \right], \tag{9}$$

where η is the learning rate. The bracketed quantity is the temporal- difference (TD) error. Repeated updates reduce the Bellman residual and move the estimate toward the Bellman fixed point.

3.4 Why a deep Q-network is required

A table cannot store a separate value for every continuous six-dimensional observation. DQN therefore uses a neural network $Q_\theta(s, a)$ to approximate all action values. For the supplied benchmark,

$$6 \longrightarrow 64 \longrightarrow 64 \longrightarrow 10,$$

with ReLU activations in the two hidden layers. The ten outputs correspond to the ten PWM commands.

For a replay sample $(s_i, a_i, r_i, s'_i, d_i)$, the target is

$$y_i = r_i + \gamma(1 - d_i) \max_{a'} Q_{\theta^-}(s'_i, a'), \tag{10}$$

and the online network is trained with a Huber TD loss

$$\mathcal{L}(\theta) = \frac{1}{B} \sum_{i=1}^{B} \text{Huber}_\delta \left(Q_\theta(s_i, a_i) - y_i \right). \tag{11}$$

3.5 The three networks used by this benchmark

The benchmark deliberately reproduces the successful MATLAB implementation and uses three copies of the same architecture:

- **online network** Q_θ: updated by Adam;
- **target network** Q_{θ^-}: generates a slowly changing Bellman target and is synchronized every 2000 environment steps;
- **behavior snapshot** Q_{θ^b}: selects greedy actions and is synchronized every 50 environment steps.

The behavior snapshot prevents every small gradient update from immediately changing the data-collection policy.

3.6 Experience replay

Each transition is stored in a circular replay buffer. Mini-batches sampled from this buffer reduce temporal correlation and reuse earlier experience. The benchmark uses a capacity of 30,000 transitions and a batch size of 512. At a domain-randomization stage boundary, the distribution changes. The code therefore clears replay, resets Adam state, synchronizes target and behavior networks, and recollects 25,000 transitions before resuming gradient updates.

3.7 ϵ-greedy exploration

The behavior policy is

$$a_k = \begin{cases} \text{uniform random action}, & \text{with probability } \epsilon_k, \\ \arg\max_a Q_{\theta^b}(s_k, a), & \text{otherwise.} \end{cases} \tag{12}$$

During the nominal stage,

$$\epsilon_k = \epsilon_{\min} + (\epsilon_0 - \epsilon_{\min}) \exp\left(-\frac{k}{\tau_\epsilon}\right), \tag{13}$$

with $\epsilon_0 = 1$, $\epsilon_{\min} = 0.001$ and $\tau_\epsilon = 400{,}000$. Exploration is partially restored after each randomization-stage transition.

4 DQN Control and Training Workflow

4.1 Environment, reward and termination

The control period is 0.005 s, corresponding to 200 Hz. Each episode is limited to 2000 steps. Angles are measured directly; velocities are formed from wrapped angle differences and filtered using

$$\dot{q}_{k,\mathrm{LPF}} = (1-\beta)\dot{q}_{k-1,\mathrm{LPF}} + \beta \dot{q}_{k,\mathrm{raw}}, \qquad \beta = 0.25. \tag{14}$$

The benchmark reward is

$$\begin{aligned} r_k = {} & 10\cos\alpha_k - 0.001\dot{\alpha}_k^2 - 0.0001\dot{\theta}_k^2 \\ & - 5\,\mathbf{1}\{|\alpha_k| > 15^\circ\}. \end{aligned} \tag{15}$$

A large positive reward therefore requires the pendulum to remain near upright with small angular velocity. The reward is a training signal; the absolute sign of an episode return is less important than its trend, stability and relation to the success metric.

4.2 Benchmark training parameters

Table 1: Main benchmark DQN parameters.

Parameter	Benchmark value
Network	$6 \to 64 \to 64 \to 10$, ReLU
Weight initialization	$\mathcal{N}(0, 0.01^2)$, zero biases
Optimizer	Adam, learning rate 5×10^{-4}
Replay capacity / warm-up	30,000 / 2000 transitions
Batch size	512
Discount factor	$\gamma = 0.95$
Training frequency	every 4 environment steps
Gradient steps	4 updates per training event
Target synchronization	every 2000 steps
Behavior synchronization	every 50 steps
Loss	Huber loss, $\delta = 1$
Double DQN	disabled; vanilla target-network maximum
Total training length	5,000,000 environment steps

4.3 Nominal training followed by domain randomization

The first two million steps reproduce the clean MATLAB-aligned nominal problem. No measurement noise, delay or residual network is active. Randomization is introduced only after the policy has had sufficient opportunity to learn swing-up and balance.

Table 2: Fixed training stages.

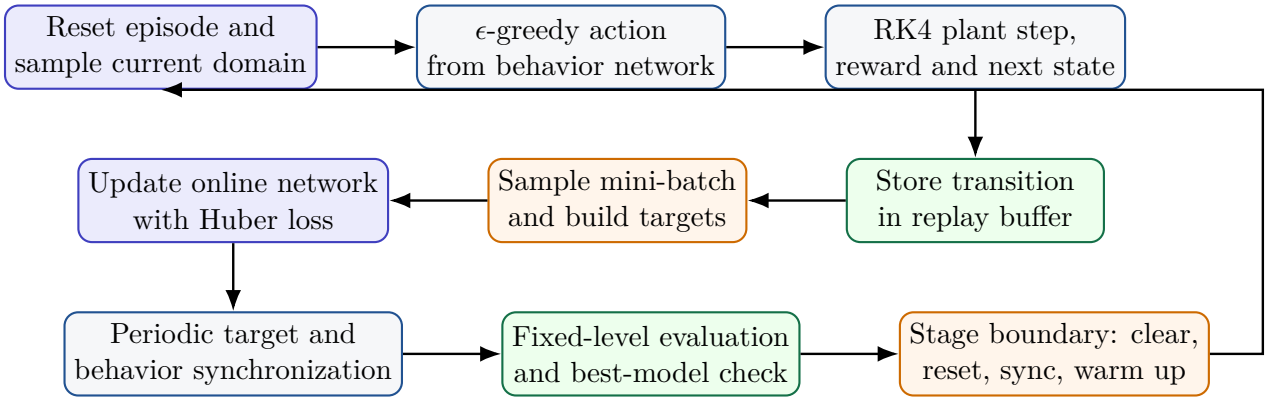
Stage	Step range	Duration	Randomization level
nominal	0–2,000,000	2,000,000	0.00
dr_0.10	2,000,000–3,000,000	1,000,000	0.10
dr_0.30	3,000,000–4,000,000	1,000,000	0.30
dr_0.50	4,000,000–5,000,000	1,000,000	0.50

A level interpolates each configured parameter range toward the nominal value. For example, level 0 is exactly nominal and level 1 would use the full configured range. The package randomizes physical parameters, initial-state spread, actuator behavior, sensor bias/noise, encoder quantization and velocity-filter settings.

IMPORTANT

A high reward before two million steps does not guarantee that the policy will survive a distribution change. Policy collapse after randomization begins is a central experimental issue in this laboratory. Do not hide collapse by plotting only the best interval. Report the full curve.

4.4 Complete online training loop



4.5 Best-model rule and direct deployment

Nominal and randomized best models are tracked separately. A new evaluation replaces the current best if:

1. its stable success rate is higher; or
2. success rate is exactly tied and mean reward per step improves by more than 0.02.

A randomized model becomes the default deployment model only if its stable success rate reaches 0.60. Otherwise the nominal best is selected. The final priority is:

1. qualified randomized best;
2. nominal best;
3. nominal two-million-step checkpoint;
4. final training state.

The trained network is already deployment-sized. There is no distillation stage. The selected network is written directly as

`selected_best_model.pt` and `deploy/model_weights.h`.

5 Using the DQN Training Package

5.1 Package structure

Path	Purpose
`run.py`	Single launcher for panel, training, smoke test and deterministic test.
`config.py`	All environment, DQN, reward, evaluation, randomization and panel parameters.
`panel.py`	Graphical control panel and live training curves.
`train_worker.py`	MATLAB-aligned custom DQN training loop.
`runtime.py`	Nonlinear Furuta environment, replay, networks, evaluation and export.
`test_worker.py`	Deterministic 30-second test of a saved run.
`stable_baselines3/`	Bundled local library; retained for project compatibility.
`third_party/`	Bundled local Gymnasium and cloudpickle dependencies.
`reference/RN_DQN_original.m`	Original MATLAB reference implementation.

5.2 Python environment and local dependencies

The launcher prioritizes the package-local copies of Gymnasium, cloudpickle and Stable-Baselines3. The active Python environment must provide PyTorch, NumPy, Matplotlib and Tkinter.

A typical launch sequence is:

```
cd dqn
python3 run.py
```

When a specific virtual environment is required, call its Python executable directly:

```
/path/to/venv/bin/python run.py
```

The terminal should print the resolved dependency paths. Confirm that the reported Gymnasium, cloudpickle and SB3 paths are inside the project directory.

5.3 Panel tabs and controls

The panel contains four tabs:

- **Common Parameters**: frequently tuned values grouped by topic;
- **All Config Parameters**: every editable value in `config.py`;
- **Training, Testing, and Curves**: training controls, stage status, reward curve, fixed-level evaluation curve and stable-success curve;
- **Full Log**: complete worker output and error messages.

The main controls are:

- **Save Common Parameters** or **Save All Parameter Changes**;
- **Start Full Training**;
- **Run Full Smoke Test**;
- **Select Existing run_dir**;
- **Start 30-Second Test**;
- **Emergency Stop Current Process**.

IMPORTANT

Always save parameter edits before starting a run. Do not edit `config.py` while a training process is active. Use the smoke test after changing code, but do not interpret a 512-step smoke test as evidence of control performance.

5.4 Recommended training workflow

1. Copy the original package to a new experiment directory. Keep an untouched benchmark copy.
2. Start the panel with `python3 run.py`.
3. Run the smoke test once to verify imports, model construction, replay dimensions, stage transition, model export and short deterministic test.
4. Choose one hypothesis, for example replay capacity, learning rate, exploration schedule, target interval or stage allocation.
5. Change only the parameters needed to test that hypothesis.
6. Save the configuration and start full training.
7. Do not interrupt a run simply because the early curve looks poor. DQN training can contain long plateaus and temporary regressions.
8. Preserve the entire run directory. Do not overwrite or rename its internal model files.
9. Compare full training curves, fixed-level evaluation curves, success rate and stage-transition behavior.
10. Record a conclusion before beginning the next parameter set.

Command-line alternatives are:

```
python3 run.py --worker train
python3 run.py --worker smoke
python3 run.py --worker test --run-dir runs/dqn_YYYYMMDD_HHMMSS --variant current
```

5.5 Run-directory outputs

Each run is written to

`runs/dqn_YYYYMMDD_HHMMSS/`.

Output	Meaning
`config_snapshot.json`	Exact configuration used by the run.
`stage_schedule.json`	Stage names, levels and number of steps.
`episode_history.csv`	Per-episode return, reward per step, length, capture and stable-run statistics.
`eval_history.csv`, `eval_history.json`	Fixed-level evaluation results.
`dr_audit.csv`	Domain samples recorded during training.
`checkpoints/model_XXXXXXXXX.pt`	Periodic checkpoints.
`best_nominal_model/best_model.pt`	Best nominal-phase checkpoint.
`best_randomized_model/best_model.pt`	Best randomized-phase checkpoint.
`recovery_model/nominal_2m_last.pt`	State saved at the end of the two-million-step nominal phase.
`final_model.pt`	Final network state, retained for traceability.

Output	Meaning
`selected_best_model.pt`	Model selected for testing and deployment.
`deploy/model_weights.h`	Direct C export of the selected network.
`test_result_current.json`	Summary of the panel/CLI deterministic test.
`test_trace_current.csv`	Step-by-step deterministic test trace.
`test_trace_current.png`	Test plot of states, velocities, PWM and reward.

5.6 How to tune without losing experimental meaning

The benchmark is intentionally not presented as the best possible setting. Students may investigate:

- learning rate and Adam settings;
- replay capacity and warm-up length;
- batch size;
- target and behavior synchronization intervals;
- exploration decay and randomized-stage exploration reset;
- train frequency and number of gradient steps;
- network width, while respecting deployment memory and timing;
- reward weights and success criteria;
- stage levels, durations and transition warm-up;
- randomization ranges, provided changes are clearly documented.

STUDENT TASK

For every parameter set, save the complete run directory and maintain a separate experiment table containing: run ID, hypothesis, changed parameters, random seed, training duration, best nominal metrics, best randomized metrics, collapse or recovery observations, selected model path, and your conclusion.

6 Using the Deployment and Test Package

6.1 Files and dependencies

File	Purpose
`rip_dqn_sim_test.py`	Nonlinear 200 Hz digital-twin panel for a .pt checkpoint, C header or full run directory.
`rip_dqn_hardware.py`	Physical-system panel, model upload, calibration, experiment control and logging.
`rip_dqn_hardware/rip_dqn_hardware.ino`	STM32/Arduino firmware for 200 Hz DQN inference and telemetry.

Install the required host packages in the active environment:

```
python3 -m pip install numpy matplotlib PyQt5 pyserial
```

PyTorch is additionally required when loading a `.pt` checkpoint. It is not required when the selected model is only a generated C header.

6.2 Digital-twin simulation panel

Launch the panel from the deployment package:

```
python3 rip_dqn_sim_test.py
```

The model selector accepts:

- a specific `.pt` checkpoint;
- a generated `model_weights.h`;
- a training run directory.

When a run directory is selected, the loader first searches for `selected_best_model.pt`, followed by the nominal best, recovery checkpoint and final model. If several ambiguous files remain, select the intended file directly.

Digital-twin workflow:

1. Click **Choose Model File** or **Choose Run Folder**.
2. Select the initial condition: MATLAB random downward, exact downward, near upright $+8°$ or near upright $-8°$.
3. Set duration, playback speed, domain-randomization level and seed.
4. Select whether to generate a CSV log and choose the output directory.
5. Click **SAVE / Load Model & Apply Settings**.
6. Confirm the displayed architecture, action set, velocity LPF and model digest.
7. Click **GO**. Use **STOP** only when necessary and **RESET** before changing to another controlled trial.
8. Open **Show Last Result Curves** and preserve the generated PNG and CSV files.

For a fair ten-run comparison, keep model, duration, initial-condition rule, randomization level and all panel settings fixed. Change only the seed according to a predefined list and record that list in the report.

6.3 Preparing the STM32 firmware

Open

`rip_dqn_hardware/rip_dqn_hardware.ino`

in the Arduino IDE, select the correct STM32 board and programmer/port settings, and upload the firmware. The firmware executes the $6 \to 64 \to 64 \to 10$ network at 200 Hz. The host panel transfers the selected model to STM32 RAM using an acknowledged, CRC-checked serial protocol.

The firmware uses the same difference-plus-LPF velocity estimate as training at large angles. Near upright it can initialize and blend toward a fixed discrete Luenberger observer. This estimator transition is configured from the hardware panel.

DANGER

Hardware experiments involve a rapidly moving rotary arm and pendulum. Clear the workspace, secure the base, keep hands and loose objects outside the motion volume, verify the emergency-stop procedure, and begin with a conservative PWM limit. Never press GO while another person is touching the mechanism.

6.4 Physical-system panel workflow

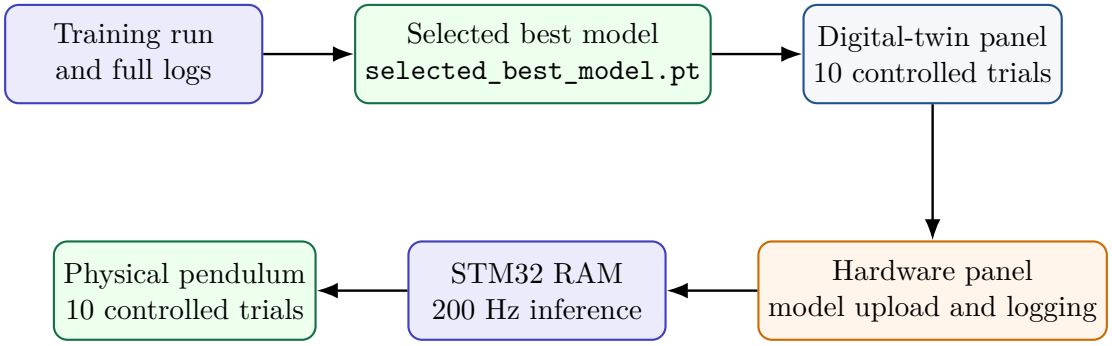
Launch:

```
python3 rip_dqn_hardware.py
```

Perform the following sequence:

1. **Connect.** Select the STM32 serial port, click **Connect**, and wait for the firmware handshake.
2. **Select the model.** Choose a `.pt` checkpoint, `model_weights.h`, or a complete training run directory.
3. **Set controller parameters.** Enter duration, PWM maximum, observer enter/exit angles, observer blend coefficient and large-angle velocity LPF coefficient.
4. **Calibrate the pendulum.** Hold the pendulum upright and click **Hold upright -> Calibrate** $\alpha = 0$. Then let it hang downward and click **Hang down -> Calibrate** $\alpha = \pi$.
5. **Verify rotary-arm scaling and motor sign.** Confirm encoder radians per count and the motor direction before a full-power run.
6. **Upload.** Click **SAVE / Upload Model & Parameters to STM32**. Wait until the model upload and configuration are acknowledged.
7. **Run.** Clear the workspace, click **GO**, and watch live angles, PWM, controller mode, observer blend, action index and maximum Q value.
8. **Stop if required.** Use **STOP** immediately if the arm approaches a mechanical hazard, cables wind around the shaft, or behavior is clearly unstable.
9. **Preserve results.** The panel saves a PNG and, when enabled, a CSV file. Record model digest, calibration values and panel parameters for every trial.

6.5 Deployment path



7 Required Experimental Work

7.1 Benchmark and improvement objective

The supplied training package is the instructor benchmark. Students must search for settings that improve at least one of the following while avoiding serious regressions in the others:

- smoother episode-reward and evaluation-reward curves;
- less severe policy collapse after reaching a good policy;
- faster or more repeatable attainment of stable success;
- better retention of nominal performance;
- successful continuation through the post-two-million-step randomized stages without collapse;
- improved fixed-level randomized evaluation.

The assignment does **not** require the student's physical-system result to be better than the instructor's result. The main requirement is a defensible, well-documented training improvement supported by complete logs and repeated experiments.

7.2 Training-stage requirement

Students must:

1. perform repeated, patient hyperparameter experiments;
2. change parameters systematically rather than randomly changing many values at once;
3. preserve the exact configuration and full training log for every run;
4. retain failed and collapsed runs as evidence, not delete them;
5. identify the best model using both reward and stable-success behavior;
6. explain what happened at the two-million-step randomization boundary;
7. state whether the final selected model came from nominal or randomized training and why.

IMPORTANT

Read the source code and this manual carefully before asking operational questions. The teaching assistants are not responsible for line-by-line code explanation or for choosing hyperparameters on behalf of students. Questions should identify the exact file, function, parameter, error message and attempted solution.

7.3 Ten simulation trials

Run the selected model ten times in `rip_dqn_sim_test.py`. Use a predefined seed list and identical settings for all ten trials. For every trial, save:

- PNG response plot;
- CSV trace;
- model path and digest;
- seed, initial-condition rule, randomization level and duration;
- whether upright capture and final stability were achieved;
- swing-up/capture time;
- stable-stage mean and standard deviation of $|\alpha|$;
- stable-stage mean and standard deviation of |PWM|;
- maximum $|\theta|$ and any termination event.

Summarize the ten trials with mean, standard deviation, minimum, maximum and success rate. Do not present only the best trial.

7.4 Ten physical-system trials

Use the parameter set and model that the student judges to be best. Perform ten physical runs under the same calibration and panel settings. Save the PNG and CSV from every run and record:

- date/time and trial number;
- model path and digest;
- upright and hanging potentiometer calibration values;
- encoder scale, motor sign, PWM limit and observer settings;
- capture success, capture time and loss-of-balance events;
- stable-stage $|\alpha|$ and |PWM| statistics;
- unusual sounds, saturation, cable interference or emergency stop.

If a trial is stopped for safety, it still counts as a trial and must be reported.

8 Required Submission and Analysis Template

8.1 Experiment table

For each training run, complete a table with at least the following columns:

Run ID	Hypothesis	Changed parameters	Nominal best	Randomized best	Conclusion

8.2 Required figures

Include:

1. full episode reward-per-step curve for the benchmark;
2. full episode reward-per-step curve for the selected student model;
3. fixed-level evaluation reward and stable-success curves;
4. a marked vertical line at every stage transition;
5. an example collapse case and an analysis of its likely cause;
6. ten-run simulation summary plots;
7. ten-run hardware summary plots;
8. one representative simulation response and one representative hardware response, selected by a stated rule rather than subjective preference.

8.3 Required discussion questions

Answer the following:

1. How does the Bellman target connect immediate reward to future control performance?
2. Why are replay, target and behavior networks all used in this package?
3. Which parameter changes improved stability, and what evidence supports that conclusion?
4. Did a better nominal model remain better after randomization?
5. What happened immediately after replay was cleared at each stage change?
6. Was the final deployment model selected from nominal or randomized training?
7. How different were simulation and physical results across ten trials?
8. Which mismatch is most likely to limit sim-to-real performance: dynamics, sensing, velocity estimation, actuator response, delay or calibration?
9. What would you change next, and what controlled experiment would test it?

9 Troubleshooting

Symptom	Check
Panel does not start	Use the intended virtual-environment Python; verify PyTorch, NumPy, Matplotlib and Tkinter.
Local dependency paths are wrong	Start from the package root; confirm printed paths point to local `third_party` and `stable_baselines3`.
Invalid device string	Inspect `RUN.device`; valid values are `auto`, `cpu`, `mps` and `cuda` without nested quotation marks.
Smoke test passes but training fails	Read the full log; smoke verifies code paths, not long-run numerical stability.
Reward improves then collapses	Compare replay composition, exploration, target/behavior synchronization, learning rate and the randomization boundary.
Model folder is ambiguous in test panel	Select `selected_best_model.pt` directly.
Hardware SAVE is disabled	Connect to firmware and wait for handshake; ensure no run or upload is active.
GO is disabled	Model and configuration must both be acknowledged after SAVE.
Model upload fails	Check serial port, firmware protocol version, cable quality and that no other program owns the port.
Hardware motion sign is wrong	Stop immediately and verify motor sign, encoder direction and calibration.

10 Summary

This laboratory connects Bellman optimality, Q-learning and deep neural-network function approximation to a complete nonlinear control experiment. Students use a MATLAB-aligned DQN benchmark, improve its training behavior through controlled experimentation, preserve all evidence, validate the selected model in ten simulation trials, and perform ten physical-system trials. The central learning outcome is not a single fortunate model; it is the ability to design, execute, record and defend a reproducible RL control experiment.