

Reinforcement Learning for Motion Control in Embodied AI: A Complete Tutorial

Motion Tracking, Residual RL, Teacher–Student Learning, Universal Control, and Sim-to-Real

2026-10-19

Contents

1. Scope and Taxonomy of Motion-Control RL	2
2. States, Commands, and Actions for Motion Control	5
3. Methods: PPO, SAC, Imitation, and Residual Learning	8
4. Motion Tracking and Physics-Based Imitation RL	8
5. HOVER, BeyondMimic, SONIC, and Universal Controllers	10
6. Privileged Teachers, Distillation, and Dynamics Adaptation	11
7. Residual RL, Closed-Loop Recovery, and Safety	13
8. Case Study: Legged Velocity Control and Coordination	13
9. Case Study: Manipulators, Dexterous Hands, and Contact	14
10. Robust Control Learning and Sim-to-Real	16
11. Evaluation of Motion, Generalization, and Recovery	18
12. Paper Roadmap, Tool Index, and Research Directions	19
References and Open-Source Resources	21

[Course home](#) · [简体中文](#) · [Simulation and RL tools](#)

Item	Details
Scope	Algorithms and control methods (What to Learn); see separate materials for PPO/SAC derivations and simulator operations
Core question	How can we train stable, generalizable, recoverable robot motion policies with versatile command interfaces?
Documentation review	2026-10-19
License	CC BY-NC-SA 4.0 · Xiong Zeng and contributors

1. Scope and Taxonomy of Motion-Control RL

Reading note: Example rewards, rates, and task settings illustrate design principles; they are not validated safe hardware parameters. Paper and tool links are clickable.

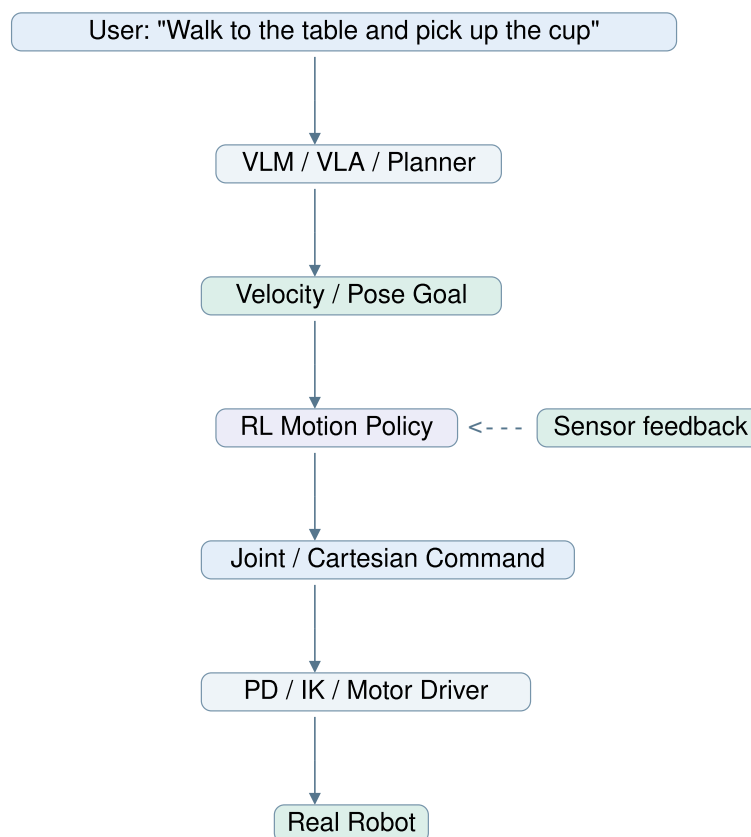
Course knowledge map: Reference motion/goal command → RL policy → safeguarded actuator → robot feedback → error correction.

1.1 Where motion-control RL fits in an embodied system

A robotic system can be approximately organized into three levels:

Level	Main responsibility	Common techniques
High level: task planning	Decide what the robot should do	LLMs, VLMs, VLAs, task planning
Middle level: motion and skill control	Decide how the robot should move	Motion planning, motion tracking, RL
Low level: actuator control	Execute motor-level commands	PID, PD, impedance and current control

Motion-control RL is mostly applied at the middle level, although it may also learn parts of low-level control. For example, a high-level system may request “walk forward at 0.5 m/s.” The motion policy produces joint commands, and a low-level controller executes them.



A goal-conditioned motion policy can be written as

$$a_t = \pi_\theta(o_t, c_t),$$

where o_t contains observations such as joint states, IMU readings, and vision; c_t is a desired velocity, pose, or skill command;

a_t is a joint or Cartesian action; and π_θ is the learned policy.

Conventional controllers typically employ explicitly designed control laws, whereas motion-control RL learns a feedback policy from interactions with a dynamical system. These approaches are complementary: practical robotic systems often combine RL with PD, inverse kinematics (IK), model-predictive control (MPC), and whole-body control (WBC).

1.2 Motion-control RL versus RL post-training of embodied foundation models

Dimension	Motion-control RL	RL post-training for foundation policies
Primary objective	Learn stable, flexible motor skills	Improve an existing policy's task performance
Typical inputs	Joints, IMU, goals, other robot states	Images, language, robot states
Typical outputs	Joint or end-effector control commands	Action chunks or action distributions
Typical models	MLPs, GRUs, Transformers	VLA, diffusion policies, flow-matching models
Methods	PPO, SAC, imitation-based RL	PPO, SAC, offline RL, specialized fine-tuning
Main interaction source	Large-scale simulation	Pretraining data, simulation, real interaction
Typical tasks	Walking, jumping, stairs, dexterity	Visuolinguistic manipulation tasks
Main research issues	Stability, robustness, motor performance	Generalization, sample efficiency, adaptation

For instance, training a Unitree G1 to walk under velocity commands is motion-control RL. Improving the grasping success rate of an already pretrained $\pi_{0.5}$ policy through RL is foundation-policy post-training. The boundary is not absolute: both may optimize the same robot actions for manipulation, but differ in architecture, initialization, and objectives.

1.3 Classification by robotic task

Category	Typical tasks	Main challenges
Legged locomotion	Walking, running, turning	Balance and contact stability
Agile locomotion	Jumping, stair climbing, obstacles, rolls	Dynamic contact and nonlinear dynamics
Whole-body motion tracking	Dance, motion imitation, trajectory tracking	Whole-body coordination and fidelity
Manipulation control	Grasping, carrying, insertion	Contacts, accuracy, force control
Dexterous manipulation	In-hand rotation, regrasping, tool use	High-dimensional actions, multi-contact dynamics
Loco-manipulation	Walking while grasping, carrying, door opening	Coordination between locomotion and manipulation

These categories overlap. A humanoid carrying a box may simultaneously require locomotion, whole-body coordination, and manipulation.

A. Legged locomotion

The objective is to move stably under a command. A common velocity command is

$$c_t = (v_x^{\text{cmd}}, v_y^{\text{cmd}}, \omega_z^{\text{cmd}}),$$

representing longitudinal, lateral, and yaw velocities. The policy outputs joint actions that track these commands.

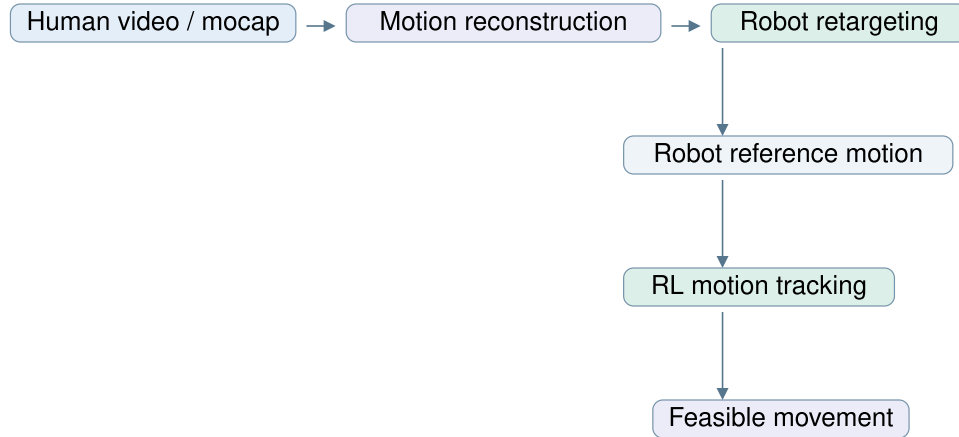
Quadruped and bipedal locomotion have developed relatively mature large-scale simulation workflows. A representative example is *Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning*, which trained and deployed an ANYmal controller using massively parallel simulation [1].

B. Agile locomotion

Agile locomotion includes jumping, stair climbing, obstacle crossing, running, fall recovery, and mid-air posture control. Compared with basic walking, the robot must exploit its dynamics and contact transitions. Climbing stairs, for example, requires coordinated foot clearance, landing timing, and center-of-mass transfer.

C. Whole-body motion tracking

The robot is given a reference motion $q^{\text{ref}}(t)$ and learns to track it while remaining dynamically feasible.



Representative approaches include DeepMimic, AMP, HOVER, and BeyondMimic. They explore reference imitation, motion-prior learning, multimodal whole-body control, and difficult humanoid motions [2][3][4].

D. Manipulation control

The goal is to pick up, carry, insert, or otherwise manipulate objects. RL can output joint commands or end-effector pose increments; an IK, operational-space control (OSC), or joint controller may then convert these into actuator commands. Compared with locomotion, these tasks emphasize end-effector accuracy, interaction forces, task success, collision safety, and recovery.

E. Dexterous manipulation

A multi-fingered hand controls several contacts simultaneously:

$$a_t = [a_t^{\text{finger1}}, a_t^{\text{finger2}}, \dots, a_t^{\text{fingerN}}]^\top.$$

Tasks include stable grasps, in-hand rotation, regrasping, tool use, and recovery after object slip. The primary difficulties are high-dimensional actions and complex finger–object contact dynamics.

F. Loco-manipulation

Loco-manipulation combines moving the robot and interacting with objects. Examples include carrying while walking, using a hand for support during obstacle crossing, manipulating objects with both hands while moving, and opening a door. The objective shifts from isolated skills to coordinated whole-body behavior and contact planning.

1.4 Classification by training strategy

Strategy	Main idea	Suitable applications
RL from scratch	Explore directly with task rewards	Basic balance and walking
Imitation RL	Use rewards defined by reference motions	Dance and motion reproduction
Residual RL	Learn corrections to a baseline controller	Trajectory repair and contact compensation
Hierarchical RL	High-level skill selection and low-level control	Multistage tasks
Teacher–student learning	Train a privileged teacher and distill it	Vision-based control and sim-to-real
Adaptive RL	Adapt through historical observations	Changing friction, payloads, dynamics

These strategies can be combined. For instance, train a privileged teacher with imitation-guided RL, distill its behavior into a student that only sees deployable sensors, and transfer that student to hardware.

2. States, Commands, and Actions for Motion Control

The core of a training environment is

$$\text{Environment} = \{\text{Observation}, \text{Action}, \text{Reward}, \text{Reset}, \text{Dynamics}\}.$$

Goal-conditioned tasks also require a Command or Goal definition. In practice, environment design can matter more than minor changes to a PPO network.

2.1 Observation: what can the robot sense?

Observation	Meaning	Typical source
Joint position	Joint angles or positions	Encoders
Joint velocity	Joint speeds	Encoders / estimators
Base angular velocity	Body rotational velocity	IMU
Projected gravity	Gravity direction expressed in body frame	Orientation estimator
Previous action	Previous control command	Controller
Velocity command	Requested robot velocity	High-level controller
Contact state	Foot / fingertip contact estimates	Contact estimator
Height map	Terrain elevation	Depth camera or other sensors

A representative proprioceptive observation is

$$o_t = [q_t, \dot{q}_t, \omega_t, g_t, a_{t-1}, c_t].$$

Always distinguish **states available in simulation** from **observations accessible on the deployed robot**. Base linear velocity, contacts, and exact object poses often require estimators on hardware.

Observation history

Motion control is frequently partially observable. Current joint angles alone do not reveal a previous slip, an external push, or actuator delay. One solution is to use a history window:

$$o_t^{\text{hist}} = [o_{t-k}, \dots, o_{t-1}, o_t].$$

An MLP, GRU, or Transformer can process this history and help the policy infer hidden dynamical and contact properties.

2.2 Action: what should RL output?

Action space	Output	Typical characteristics
Joint position	Target joint positions	A stable and accessible baseline
Joint velocity	Target joint velocities	Suitable for velocity-control interfaces
Joint torque	Target joint torques	Direct control; harder training and safety challenges
Cartesian action	End-effector pose or velocity	Often natural for robot arms

A. Joint-position control

A common parameterization is

$$q_t^{\text{target}} = q^{\text{default}} + s_a a_t, \quad a_t = \pi_\theta(o_t).$$

A low-level PD controller then computes

$$\tau_t = K_p(q_t^{\text{target}} - q_t) - K_d \dot{q}_t.$$

For reference tracking, one can instead use

$$q_t^{\text{target}} = q_t^{\text{ref}} + s_a a_t.$$

This is a residual action around the reference. Using PD control relieves the learned policy from modeling every aspect of actuator control.

B. Joint-torque control

The policy directly predicts $\tau_t = \pi_\theta(o_t)$. This gives greater direct control authority, but current, torque, temperature, speed, and position constraints must be enforced explicitly.

C. Cartesian control

For a robot arm, an action may be $a_t = [\Delta x, \Delta y, \Delta z, \Delta R]$. An IK or OSC module maps the Cartesian command into joint commands.



The orientation increment ΔR can be represented using a rotation vector or another consistent parameterization. Isaac Lab offers several joint-space and Cartesian control interfaces [6].

Suggested starting points

Robot	Initial action-space recommendation
Quadruped / biped	Joint-position targets
Humanoid motion tracking	Joint-position targets / residuals
Conventional robot arm	Cartesian pose increments
Multi-fingered hand	Finger-joint position targets
Arm plus dexterous hand	Cartesian arm action plus finger-joint action

A policy's output need not match the final actuator interface. For example, the policy may predict a Cartesian pose change, while the motor drivers execute joint-position commands.

2.3 Reward: what behavior should the robot optimize?

A useful decomposition is

$$r_t = r_t^{\text{task}} + r_t^{\text{stability}} - r_t^{\text{regularization}}.$$

For bipedal velocity tracking, possible terms are

$$r_{\text{velocity}} = \exp\left(-\frac{\|v_t - v_t^{\text{cmd}}\|^2}{\sigma_v^2}\right),$$

$$r_{\text{orientation}} = \exp\left(-\frac{\|e_t^{\text{orientation}}\|^2}{\sigma_R^2}\right),$$

$$r_{\text{smooth}} = -\|a_t - a_{t-1}\|^2.$$

One combined reward is

$$r_t = w_v r_{\text{velocity}} + w_R r_{\text{orientation}} - w_a \|a_t - a_{t-1}\|^2 - w_\tau \|\tau_t\|^2.$$

Actual tasks may add body-height, foot-slip, collision, joint-limit, energy, and contact-stability terms. Rewarding only forward velocity may encourage lunging, implausibly fast leg motion, or exploitation of simulator artifacts. Conversely, overly strong regularization may cause the robot to remain still.

Principle: rewards should capture both task accomplishment and safe, stable, deployable behavior.

2.4 Reset: which initial states should be sampled?

Simple locomotion may start from a standing pose. Motion tracking and multistage manipulation should not always begin from one fixed state, or the policy may fail to recover when entering a task midway.

Uniform Reference State Initialization (Uniform RSI): sample a time from a reference motion,

$$t_0 \sim U(0, T),$$

then initialize the robot near the corresponding reference state.

Event-biased RSI: increase the sampling probability near takeoff, landing, contact switches, grasp initiation, or release. Ensure sampled states are physically and contact consistent; unrealistic resets can make learning harder rather than easier.

3. Methods: PPO, SAC, Imitation, and Residual Learning

This tutorial retains only the **control-method selection** aspects of PPO and SAC. Their derivations (GAE, actor/critic updates, Q-learning and entropy objectives) belong in the dedicated PPO/SAC notes.

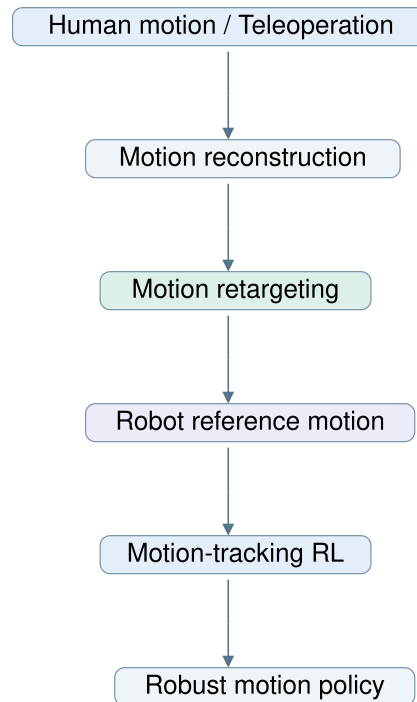
Method	Typical motion-control role	Main limitation
PPO	Large-scale simulation, motion tracking and contact-rich exploration	On-policy; limited reuse of earlier trajectories
SAC	Continuous actions, expensive interaction and replay-based adaptation	Critic/extrapolation error; sparse outcomes and async vectorization complicate implementation
Imitation + RL	Kinematic references exist but direct replay is physically infeasible	Retargeting and contact mismatch; incomplete reference-state coverage
Residual RL	An existing PD/IK/MPC/imitation controller needs robustness improvements	Base-controller limitations remain; residual authority needs bounds
Teacher–Student	Simulation has privileged state; hardware has only history/vision/proprioception	Partial observability, distribution shift and compounding errors

Write the actor as $a_t = \pi_{\theta}(o_{\leq t}, c_t)$. In safety-critical motion, actuators receive a constrained command $u_t = \text{Shield}(g(a_t), x_t)$, not an unrestricted neural output. Simulator/trainer interfacing is handled by the companion tools guide.

4. Motion Tracking and Physics-Based Imitation RL

Direct exploration can be difficult for jumping, dance, gymnastics, and high-dimensional dexterous motions. Reference motions provide useful guidance.

4.1 The basic pipeline



Reference data may originate from motion capture, human-video reconstruction, teleoperation, existing controllers, or motion planners.

4.2 Motion-tracking rewards

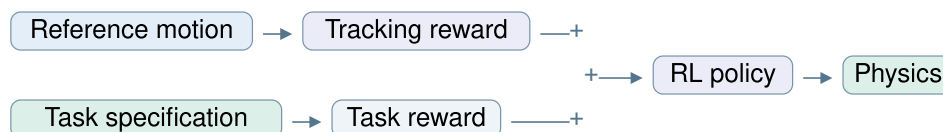
For reference joint positions q_t^{ref} , a simple reward is

$$r_{\text{track}} = \exp\left(-\frac{\|q_t - q_t^{\text{ref}}\|^2}{\sigma_q^2}\right).$$

Whole-body tasks may also track root position, root orientation, joint velocity, end-effector positions, and contact schedules. Similar joint angles alone do not guarantee a physically stable movement. The policy must balance **motion fidelity**, **dynamic feasibility**, and **stability**.

4.3 DeepMimic

DeepMimic is a classic example of combining reference-motion imitation rewards with task rewards to train physical characters to perform complex actions [2].



Project: <https://xbpeng.github.io/projects/DeepMimic/>

4.4 AMP: Adversarial Motion Priors

Conventional motion tracking follows a specified reference at each time step. Adversarial Motion Priors (AMP) instead train a discriminator to provide a motion-style reward from reference datasets. A policy can then accomplish its task while exhibiting motions consistent with that prior, such as a natural-looking gait, without strictly reproducing one reference frame by frame [3].

4.5 Motion tracking versus AMP

Property	Motion-tracking RL	AMP
Requires motion data	Typically reference trajectories	Motion examples
Explicitly follows a specified trajectory	Yes	Not necessarily
Reward source	Reference tracking error	Learned motion-prior reward
Primary objective	Reproduce a target motion	Match a style / motion distribution
Major challenge	Temporal alignment and stability	Discriminator training and reward balancing

4.6 From tracking one clip to universal control

Projects such as HOVER investigate unifying multiple motion-command modes within a whole-body policy so that the controller is not restricted to reproducing a fixed sequence [4].

Trend: single-motion tracking policy $\rightarrow$ multimode / universal motion controller.

4.7 Motion tracking is a closed-loop, deployable control problem

Let a reference motion contain root pose, joint angles and end-effector keypoints: $m_t^{\text{ref}} = (T_{B,t}^{\text{ref}}, q_t^{\text{ref}}, p_{k,t}^{\text{ref}})$. The policy must not merely replay q_t^{ref} . Conditioned on proprioception o_t and a reference horizon $c_t = (m_{t-h}^{\text{ref}}, \dots, m_{t+H}^{\text{ref}})$, it must compensate for dynamics and contact errors.

$$r_t = w_p \exp\left(-\frac{\|p_t - p_t^{\text{ref}}\|^2}{\sigma_p^2}\right) + w_q \exp\left(-\frac{\|q_t - q_t^{\text{ref}}\|^2}{\sigma_q^2}\right) + w_R \exp\left(-\frac{d_{SO(3)}(R_t, R_t^{\text{ref}})^2}{\sigma_R^2}\right) - \lambda_u \|u_t\|^2 - \lambda_{\Delta u} \|u_t - u_{t-1}\|^2.$$

This is an illustrative reward structure, not an exact implementation from a paper. Distinguish world-frame and root-frame errors; do not grant a deployable actor hidden global ground truth.

RSI and event sampling: Uniform reference-state initialization broadens temporal coverage. Event-biased RSI emphasizes contact creation/release, landing, object slip and dropping. Mix the two during training while keeping evaluation distributions separate.

Trajectory consistency: When randomizing a trajectory's initial SE(3) pose, apply the *same rigid transformation* to every subsequent reference pose and goal; otherwise the policy sees artificial command discontinuities.

5. HOVER, BeyondMimic, SONIC, and Universal Controllers

The distinction between a single-task policy and a universal controller is not just model size; it is whether **the command interface covers multiple goals, the training distribution covers diverse motions, and closed-loop control turns goals into physical movement.**

5.1 From tracking to general-purpose motor skills

Consider $a_t = \pi_\theta(o_{\leq t}, c_t, z_t)$, where c_t may be root velocity, head/hand poses, joint targets, full-body motion snippets, or high-level motion tokens. The latent z_t may summarize history, dynamics, or skill identity. Command modalities must share a physically consistent action space.

Representative method	Source of generality	Core lesson
DeepMimic	Reference tracking rewards and RSI	Make one reference physically feasible first
AMP	Learned motion prior / style	Distribution matching differs from exact reference tracking
HOVER	Unified multi-mode command and policy distillation	Masks select root, end-effector and joint-level commands
BeyondMimic	High-quality tracking primitives and guided diffusion	Compositional skill generation extends beyond PPO tracking
SONIC	Scaled human motion tracking and shared token interface	Connect VR, human video and VLA to low-level whole-body control

5.2 What HOVER, BeyondMimic and SONIC actually solve

HOVER unifies multiple control modes through a versatile command space and distillation. Consistent mask semantics and seamless switching are central challenges.

BeyondMimic learns high-quality tracking primitives and uses generative guidance for downstream motion synthesis. It includes both a motion-tracking controller and a higher-level generative composition mechanism; it is not merely another PPO tracker.

SONIC scales reference-motion training across data and model capacity, and exposes a shared motion-token interface to higher-level sources. Tokens are motor commands to a low-level controller, not unconstrained actuator torques.

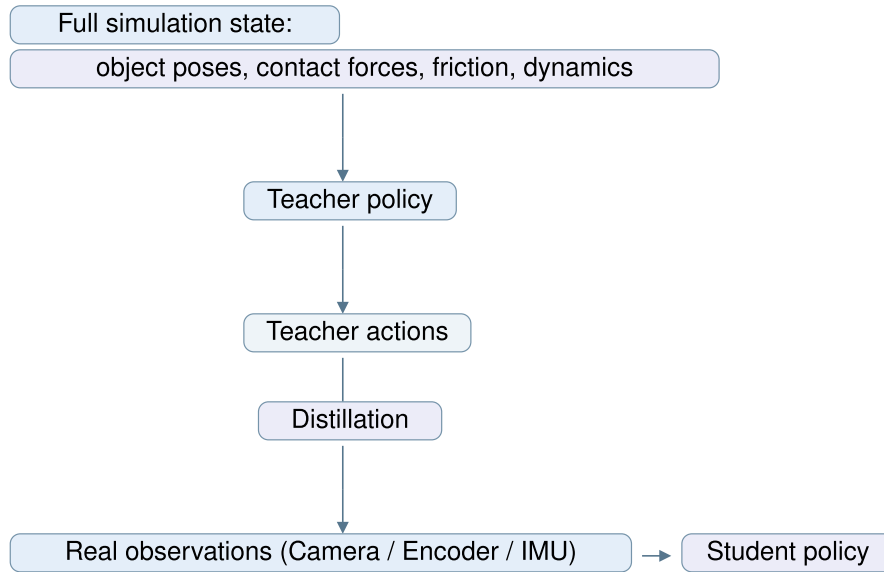
5.3 What to measure for true generality

Report unseen-motion tracking, command-mode transition robustness, perturbation recovery, coverage of body/contact states, and real-time closed-loop latency. **Low average tracking error on training motions does not establish a universal controller.**

6. Privileged Teachers, Distillation, and Dynamics Adaptation

6.1 Why use teacher–student training?

A simulator can expose exact object positions and velocities, contact forces, friction coefficients, and hidden dynamical parameters. These may be unavailable or noisy in a real robot. A policy that directly depends on them may fail to transfer.



6.2 Standard training procedure

First train a teacher with privileged state information:

$$a_t^T = \pi_T(s_t).$$

Then train a student with deployable observations only:

$$a_t^S = \pi_S(o_t).$$

A common action-distillation loss is

$$\mathcal{L}_{\text{distill}} = \mathbb{E} [\|a_t^S - a_t^T\|^2].$$

The student may use observation histories and learned state estimators to infer hidden properties. Practical distillation also needs to account for differences between the state distributions visited by teacher and student.

6.3 Asymmetric actor-critic versus teacher-student

Asymmetric actor-critic: the actor uses only deployable observations from the start of training, while the critic can use privileged simulation information.

Teacher-student: a teacher actor may itself use privileged information, and its behavior is subsequently transferred to a student actor with limited observations.

The first is a **training architecture**; the second is a **policy-transfer procedure**. They can be combined. Isaac Lab and RSL-RL support relevant observation groups and training configurations [5][6].

6.4 RMA: Rapid Motor Adaptation

Rapid Motor Adaptation (RMA) integrates a motor policy with a mechanism for adapting to unknown dynamics. Recent observation history provides information that helps the robot respond to changing terrain, friction, payload, and dynamics [7].

Paper: <https://arxiv.org/abs/2107.04034>

7. Residual RL, Closed-Loop Recovery, and Safety

7.1 Residual RL preserves a useful nominal controller

Given a nominal policy $u_t^{(0)} = \pi_0(o_t, c_t)$ (PD, IK, MPC or pretrained tracker), learn a bounded correction:

$$u_t = \text{clip}\left(u_t^{(0)} + \alpha \Delta u_\theta(o_t, c_t), u_{\min}, u_{\max}\right).$$

Corrections may act on joint targets, Cartesian increments, motion references or low-level torques. Their operating rates, reachable regions and stability assumptions differ: **editing a high-level reference is not equivalent to direct torque compensation**. To preserve any nominal-controller guarantee, enforce rate/amplitude bounds and analyze switching contacts.

7.2 Recovery from failure requires more than a reward term

A task may transition through **approach** → **contact** → **manipulate** → **slip/drop** → **re-localize** → **regrasp** → **continue**. Terminating every rollout at the first drop eliminates nearly all recovery examples from the training distribution.

- **State coverage:** Sample pre-perturbation, slipping, resting-after-drop and regraspable states; RSI around successful trajectories alone is insufficient.
- **Goal integrity:** Re-estimate local reference or object pose after failure; do not silently change success definitions.
- **Sensing:** Re-localization needs post-drop visual observations or contact/proprioceptive evidence. A purely time-indexed tracker is inadequate.
- **Policy structure:** A high-level state machine with closed-loop skills is often an effective baseline; a unified policy is possible if its recovery data and failure cost are adequate.
- **Metrics:** Report $P(\text{success} \mid \text{recoverable failure})$, regrasp count, time-to-recovery, safety events and end-to-end completion separately.

7.3 Physical safety and the limits of RL

Fast PD, impedance control, IK and WBC still handle actuator-level feedback and constraints. RL usually improves references and robust corrective behavior. For impacts, high-speed tasks or precise assembly, independently enforce joint/workspace limits, speed/torque bounds, emergency stops and supervision. Empirical rollout success is not a proof of ISS, CLF or contraction stability.

8. Case Study: Legged Velocity Control and Coordination

8.1 Task definition

Train the G1 to move stably under $c_t = [v_x^{\text{cmd}}, v_y^{\text{cmd}}, \omega_z^{\text{cmd}}]$, including forward/backward movement, lateral movement, and turning.

8.2 Observation

Start with proprioception: joint positions, joint velocities, base angular velocity, projected gravity, previous actions, and velocity commands. For stairs or uneven terrain, gradually add height scans, contact estimates, observation histories, and visual features.

8.3 Action

Use joint-position targets:

$$q_t^{\text{target}} = q^{\text{default}} + s_a \pi_\theta(o_t),$$

executed by a low-level PD controller. This makes it easier to reuse existing hardware interfaces.

8.4 Reward components

Reward	Purpose
Linear velocity tracking	Track longitudinal and lateral velocity
Angular velocity tracking	Track yaw rate
Orientation stability	Maintain appropriate body posture
Action smoothness	Reduce oscillations
Joint-limit penalty	Avoid extreme joint configurations
Foot-slip penalty	Reduce foot sliding
Termination penalty	Penalize falls and failures

8.5 Curriculum learning

Stage	Training condition	Goal
1	Standing	Basic balance
2	Slow forward walking	Basic gait
3	Forward/backward/lateral walking	Broader speed range
4	Turning	Angular-velocity tracking
5	External pushes	Robustness and recovery
6	Uneven terrain	Contact adaptation
7	Stair climbing	Challenging locomotion

The command and terrain distributions can be expanded continuously. A separate controller need not be retrained from scratch for every stage.

8.6 Open-source starting point

[Unitree RL Lab](#) builds on Isaac Lab and includes training and deployment-related code for Unitree platforms [8]. The following commands are **illustrative for the repository interface**; verify the current tasks and dependencies in the checked-out version:

First controlled study: reproduce flat-ground walking, then ablate action smoothness, domain randomization, observation history, and the foot-slip penalty. Compare convergence, velocity tracking, and perturbation recovery.

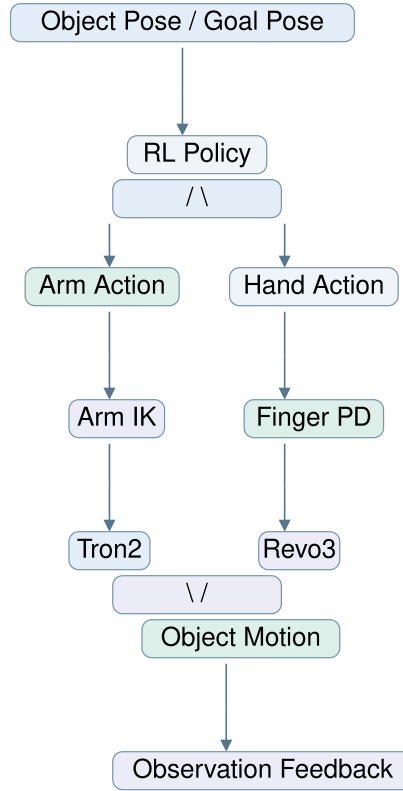
9. Case Study: Manipulators, Dexterous Hands, and Contact

Consider a **Tron2 arm**, **Revo3 dexterous hand**, and **screwdriver** to illustrate how locomotion-style RL environment design extends to contact-rich manipulation.

9.1 Task definition

1. Pick up a screwdriver from multiple positions on a tabletop.
2. Coordinate arm posture and finger contacts.
3. Move the screwdriver to a designated location.
4. Rotate it by a desired angle.
5. Maintain or recover the grasp after disturbances.

9.2 Recommended control architecture



It is reasonable to train independent arm and hand controllers first, then move toward a coupled policy.

9.3 Three training routes

Route A: RL from scratch. Define rewards for grasping, lifting, rotation, and success:

$$r_t = r_{\text{grasp}} + r_{\text{lift}} + r_{\text{rotate}} + r_{\text{success}}.$$

This does not require reference trajectories, but exploration in a high-dimensional contact problem is difficult.

Route B: motion-tracking RL. Reconstruct reference end-effector and finger trajectories, $T_{\text{hand}}^{\text{ref}}(t)$ and $q_{\text{finger}}^{\text{ref}}(t)$, from human videos or teleoperation. This reduces exploration difficulty, but one demonstration rarely covers all initial object poses and contact variations.

Route C: residual RL. Add a constrained correction to a trajectory tracker, an IK controller, or an existing policy:

$$a_t = a_t^{\text{base}} + \Delta a_t^{\text{RL}}.$$

For example, the baseline could track an arm-end-effector trajectory while RL adjusts finger contacts; alternatively, RL can produce limited corrections for both arm and hand. Residual magnitudes, action rates, and contact forces must be constrained.

9.4 Observation design

One possible observation vector is

$$o_t = [q_{\text{arm}}, q_{\text{hand}}, \dot{q}, \hat{T}_{\text{object}}, T_{\text{goal}}, a_{t-1}].$$

Here $\hat{T}_{\text{object}}$ is estimated from head- or wrist-mounted cameras. If object velocity is not directly observable, histories or a state estimator can help infer it. **Do not silently give the actor ground-truth object state that only exists in simulation.** The critic, however, may use privileged information.

9.5 Reward design

Reward	Role
Reach	Move the hand near the target
Grasp	Establish a stable grasp
Lift	Lift the object
Position	Move the object to the target region
Rotation	Achieve the desired rotation
Stability	Reduce slip and drops
Smoothness	Avoid action oscillations
Success	Reward full task completion

A policy with only a terminal rotation-success reward may rarely discover useful grasps through random exploration. A curriculum and carefully chosen intermediate rewards often help.

9.6 Proposed training stages

Stage	Task	Goal
1	Maintain a pre-established grasp	Basic contact control
2	Rotate an already-held object	In-hand manipulation
3	Reach and grasp	Grasp acquisition
4	Grasp, then rotate	Multistage operation
5	Randomize initial object pose	Generalization
6	Add object-pose estimation errors	Visual robustness
7	Add external disturbances	Recovery
8	Evaluate real-robot closed-loop control	Test transfer

When reference trajectories are available, Uniform RSI and Event-biased RSI can emphasize critical segments such as contact establishment, release, and object rotation.

10. Robust Control Learning and Sim-to-Real

10.1 Why simulation success does not ensure hardware success

Mismatch	Typical cause
Mass / inertia	Errors in link mass and inertia
Friction	Incorrect ground or object friction
Actuator	Motor response, dead zones, internal controllers
Delay	Communication and control latency
Observation noise	Encoder, IMU, and visual error
Contact	Contact solver approximations
Geometry	Incorrect collision geometry

Mismatch	Typical cause
Calibration	Joint zero positions, frame transforms, extrinsics

For instance, a simulated motor may respond much faster than its physical counterpart; the resulting timing error can induce oscillations.

10.2 Domain randomization

Randomize environment and dynamics parameters during training:

$$\xi \sim p(\xi).$$

The vector ξ may include link mass, center of mass, friction, motor strength, action delay, and sensor noise.

Randomized parameter	Purpose
Mass	Robustness to mass mismatch
Center of mass	Robustness to COM errors
Friction	Robustness to contact variation
Motor strength	Robustness to actuator variation
Action delay	Robustness to latency
Observation noise	Robustness to sensor errors
External pushes	Robustness to disturbances

Ranges should reflect calibration, hardware specifications, and physically plausible uncertainty. Wider randomization is not automatically better.

10.3 Sim-to-sim evaluation

A policy trained in Isaac Lab can be tested in MuJoCo, or vice versa, to detect over-reliance on simulator-specific numerical behavior.



If the controller behaves very differently between engines, inspect dynamics, contact, and action interfaces first. Passing sim-to-sim tests is additional evidence, not a guarantee of real-world safety.

10.4 Real-robot deployment checklist

- Match observation dimensions, ordering, and normalization to training.
- Verify joint indexing, sign conventions, units, coordinate frames, and zero positions.
- Validate action scales, PD gains, control rates, and execution timing.
- Enforce position, velocity, torque, and temperature limits.
- Provide safe fallback behavior, communication watchdogs, and emergency stops.
- Use fall protection for bipeds; guard against excessive contact forces, jamming, and dropped objects for dexterous hands.
- Begin with low-risk, small-amplitude tests before expanding the task envelope.

Advanced topic: learning the simulation–reality dynamics gap

Work such as ASAP uses data from real robots to help align simulated and physical dynamics, thereby improving sim-to-real transfer of agile whole-body behavior [9].

10.5 Distinguish robust learning from engineering calibration

Mismatch	Primary layer	Sensible response
TCP definition, joint zero, camera intrinsics/extrinsics	Simulation/deployment engineering	Calibrate and verify frames; do not expect DR to erase preventable errors
Actuator delay, friction, inertia variation	Control and dynamics adaptation	System ID, DR, history/latent adaptation, bounded residuals
Pose-estimation bias or occlusion	Observation and feedback	Noise/occlusion training, temporal filtering, active sensing, re-localization
Untrained contact/drop states	State coverage and recovery	Event-biased RSI, recovery skills and goal replanning

The tools guide explains *how* to implement and verify parameters in a simulator. This chapter explains *how policies* should use feedback to retain competence under unmodeled errors. Sim-to-real is not a magic replacement for URDF validation, extrinsic calibration or communications debugging.

11. Evaluation of Motion, Generalization, and Recovery

An increasing training reward does not necessarily imply a better deployable controller. Use held-out conditions and independent task metrics.

11.1 Legged-robot metrics

Metric	Meaning
Success rate	Fraction of completed tasks
Fall rate	Frequency of falls
Velocity tracking error	Difference between actual and commanded velocity
Orientation error	Body pose deviation
Energy consumption	Power or energy cost
Foot slip	Unwanted foot sliding
Recovery rate	Success after perturbations
Generalization	Performance on unseen terrain or parameters

11.2 Motion-tracking metrics

Metric	Meaning
Joint tracking error	Joint trajectory deviation
Root position error	Base-position deviation
Root orientation error	Base-orientation deviation
End-effector error	End-effector tracking deviation
Contact consistency	Agreement with intended contacts
Motion completion rate	Fraction of completed motion clips

Metric	Meaning
Recovery rate	Robustness after disturbances

11.3 Dexterous-manipulation metrics

Metric	Meaning
Task success rate	Fraction of successful manipulation trials
Object pose error	Final or trajectory pose deviation
Grasp stability	Ability to retain a stable grasp
Object drop rate	Frequency of dropped objects
Contact slip	Relative slip at contacts
Recovery rate	Recovery after disturbance
Completion time	Time required to finish a task
Generalization	Performance under unseen object poses

Fair comparison: match the environment and task, the interaction budget, and test conditions; report multiple random seeds, unseen initial states, and metrics independent of the training reward. Keep training success, simulated generalization, and real-world performance separate.

12. Paper Roadmap, Tool Index, and Research Directions

This section lists **research-motivated starting points**, not installation procedures. Setup, vectorized backends, version compatibility and performance debugging are centralized in the companion Isaac/MuJoCo guide.

Control topic	Algorithmic aspects to inspect	Framework / paper entry
PPO velocity tracking	Rollouts, GAE, rewards and constraints	RSL-RL + Isaac Lab
Reference-motion tracking and distillation	Retargeting, RSI and DAgger	HOVER, SONIC
Dexterous, contact-rich control	Action design, contact objectives, regrasp	MuJoCo Playground / Isaac Lab
General-purpose human-motion control	Scaled tracking, composition and command abstraction	BeyondMimic, SONIC

A training platform does not mandate PPO or guarantee compatibility across all backends. When reading code, distinguish policy inputs/outputs, teacher privileges, objective functions and hardware control rates from the simulator implementation.

Paper / project	Year	Core topic	What to learn
DeepMimic	2018	Motion-imitation RL	Reference rewards and RSI
AMP	2021	Adversarial motion priors	Style rewards from motion datasets
RMA	2021	Rapid adaptation	Observation history and dynamics
Learning to Walk in Minutes	2022	Parallel PPO	Large-scale locomotion training
Learning Robust Perceptive Locomotion	2022	Perceptive locomotion	Terrain perception and transfer

Paper / project	Year	Core topic	What to learn
HOVER	2025	Multimode whole-body control	Universal command representations
ASAP	2025	Dynamics-gap alignment	Sim-to-real
BeyondMimic	2025	Complex humanoid motions	Tracking and skill generalization
MuJoCo Playground	2025	GPU robot learning	Modern experimentation platform

Four particularly useful starting points are:

1. **Learning to Walk in Minutes:** parallel PPO, reward design, curriculum learning, and sim-to-real [1].
2. **DeepMimic:** reference motion, imitation rewards, reference-state initialization, and physics-based motion tracking [2].
3. **RMA:** privileged information, observation history, motor adaptation, and deployment [7].
4. **HOVER:** whole-body control, multiple command modalities, tracking, and distillation [4].

12.1 Stage 1: basic motion-control RL

Goal: train a quadruped or biped to walk stably.

Learn Isaac Lab, RSL-RL, PPO, observation/action/reward design, and domain randomization. First reproduce velocity tracking on a platform such as G1; then ablate reward terms and observations.

12.2 Stage 2: motion-tracking RL

Goal: track a reference while maintaining dynamic feasibility and recovery ability.

Study motion retargeting, tracking rewards, RSI, curricula, and residual RL. Start from one reference motion, then vary initial states and external disturbances.

12.3 Stage 3: teacher–student and sim-to-real

Goal: eliminate dependence on simulator-only ground-truth state.

Study asymmetric actor–critic, privileged information, distillation, observation histories, and domain randomization. Train a privileged teacher, transfer its behavior to a deployable student, and test the controller in another simulator.

12.4 Stage 4: complex motion control

Direction	Example tasks
Humanoid locomotion	Stairs, obstacles, jumping, fall recovery
Dexterous manipulation	In-hand rotation, tool use, disturbance recovery

Focus on contact dynamics, task curricula, strong robustness, and closed-loop hardware control.

12.5 Stage 5: universal motion controllers

Investigate a conditional policy that can handle multiple commands:

$$a_t = \pi_\theta(o_t, c_t).$$

The condition c_t may encode desired velocity, target pose, a reference-motion segment, desired object motion, or skill identity. Related directions include goal-conditioned motion control, universal motion tracking, hierarchical motion policies,

residual motor adaptation, and learning-based whole-body control.

References and Open-Source Resources

The following links point primarily to papers, project pages, and official repositories. Confirm final bibliographic details and currently supported task IDs from the relevant publication and software release.

1. Rudin et al., *Learning to Walk in Minutes Using Massively Parallel Deep Reinforcement Learning* (2022)
2. Peng et al., *DeepMimic* (2018)
3. Peng et al., *AMP: Adversarial Motion Priors* (2021)
4. HOVER: Versatile Neural Whole-Body Controller for Humanoid Robots; paper
5. RSL-RL official repository
6. Isaac Lab official repository; environment catalog
7. RMA: Rapid Motor Adaptation for Legged Robots
8. Unitree RL Lab official repository
9. ASAP: Aligning Simulation and Real-World Physics
10. MuJoCo Playground official repository
11. legged_gym official repository
12. skrl official repository

Document version: October 15, 2026. Numerical settings are instructional examples, not validated safety configurations for real robots.

13. BeyondMimic: From Motion Tracking to Versatile Humanoid Control via Guided Diffusion
14. SONIC: Supersizing Motion Tracking for Natural Humanoid Whole-Body Control
15. SONIC official project
16. HOVER paper and multi-mode command interface

Course boundary: Robot modeling, Isaac/MuJoCo setup, actuator matching, parallel training and performance diagnostics belong in the companion simulation/tools guide. Detailed PPO/GAE and SAC/Q-learning derivations belong in the dedicated algorithm notes.