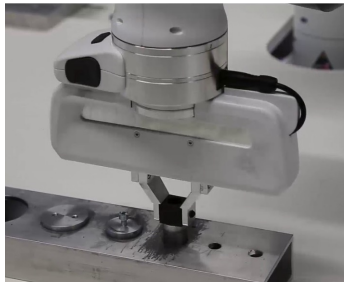
Reinforcement Learning for Robotics

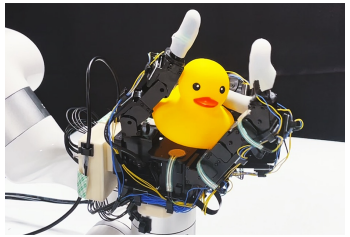
Policies, Algorithms, and Reward Engineering

Why Not Just Program the Robot?

Peg-in-hole insertion. Sub-millimeter tolerances; contact forces depend on surface condition and chamfer geometry. Impedance control handles compliant approach but cannot adapt to contact ambiguity without an accurate model for each part variant.



In-hand re-orientation. A multi-fingered hand involves 24+ contact points with changing friction and normal force distributions. No tractable analytical model exists for arbitrary object geometries.



Locomotion over unknown terrain. Ground reaction forces change with every footstep on deformable ground. A ZMP-based planner requires terrain geometry in advance and cannot adapt to unstructured environments.

Three things we lack: accurate contact models, affordable real-world data, and infinite engineering time. RL offers a different contract — specify what success looks like, and let the robot learn.

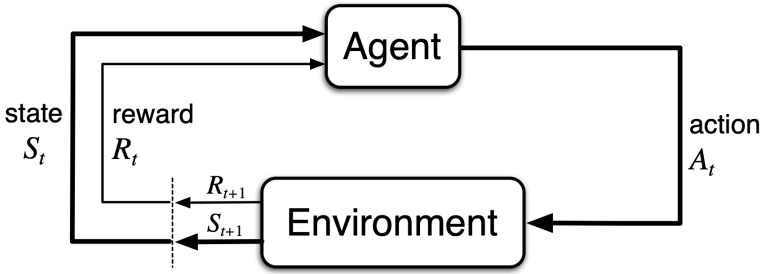
The MDP: A Robot's World Model

Symbol	Domain	Meaning
S	Set	State space
A	Set	Action space
T(s'|s,a)	S×A×S→[0,1]	Transition probability distribution
R(s,a,s')	S×A×S→ℝ	Reward function
γ	[0,1)	Discount factor

EQ1 — MDP Tuple

$$\mathcal{M} = (\mathcal{S},\ \mathcal{A},\ T,\ R,\ \gamma)$$

- $\mathcal{S}$: joint positions $q \in \mathbb{R}^7$, velocities $\dot{q} \in \mathbb{R}^7$, end-effector pose $x \in \mathrm{SE}(3)$, object pose (Lecture 5)
- $\mathcal{A}$: joint position targets $q_{\text{des}} \in \mathbb{R}^7$ passed to the low-level PD controller
- R: binary success indicator (sparse) or $-\|p_{\text{ee}} - p_{\text{goal}}\|$ (dense)
- $\gamma = 0.99$ is standard for manipulation tasks



Note: s is the ground-truth environment state; o is the robot's observation. The Markov assumption requires s to contain all future-relevant information.

Value Functions and the Bellman Equations

Symbol	Domain	Meaning
$\pi(a \mid s)$	$\mathcal{S} \to \Delta(\mathcal{A})$	Stochastic policy
G_t	$\mathbb{R}$	Discounted return from time t
$V^\pi(s)$	$\mathcal{S} \to \mathbb{R}$	State-value function under π
$Q^\pi(s,a)$	$\mathcal{S} \times \mathcal{A} \to \mathbb{R}$	Action-value function under π
$A^\pi(s,a)$	$\mathcal{S} \times \mathcal{A} \to \mathbb{R}$	Advantage: $Q^\pi(s,a) - V^\pi(s)$

EQ2 — Discounted Return

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

EQ3 — Bellman Equation

$$V^\pi(s) = \mathbb{E}_\pi[R(s,a,s') + \gamma V^\pi(s') \mid s_t = s]$$

EQ4 — Advantage Function

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

Key Insight: Why the advantage function matters

The Q-function conflates two things: how good the current state is, and how good the chosen action is. Subtracting $V^\pi(s)$ isolates the latter. This is not a technical detail — it is the core reason actor-critic methods have lower variance than raw policy gradients.

The better the critic estimates V^π , the lower the variance of the policy gradient signal the actor receives.

The Markov Assumption in Robotics

When it holds: Fully observed joint state (position, velocity, torque) in a rigid-body environment without hidden contacts — most locomotion tasks and simple pick-and-place manipulation.

When it fails: RGB observations lose depth and occlusion information; contact state is not observable from joint positions alone; actuator time delay means s_t does not fully predict s_{t+1} without recent action history.

Engineering responses: Augment the state with a history window of recent (observation, action) pairs, or use a recurrent policy (LSTM or Transformer) that maintains an internal summary.

Forward connection: World Models returns to this — world models learn a compact latent state approximating the true Markov state from non-Markovian observations.

Key Insight: When the Markov assumption breaks down

Most real-robot RL papers use proprioceptive observations — joint positions, velocities, IMU — which are approximately Markovian. Contact forces are not directly observable and introduce partial observability. The practical fix is to include a short history window of recent observations in the state input, giving the policy enough context to implicitly infer hidden contact states. It is an engineering approximation, not a theoretical solution, and it works well enough for most manipulation and locomotion tasks in practice.

RL as Optimization: What We Are Solving

Symbol	Domain	Meaning
θ	$\mathbb{R}^d$	Policy parameters (neural network weights)
π_θ		Policy parameterized by θ
$J(\theta)$	$\mathbb{R}$	Expected discounted return (performance objective)
τ		Trajectory $(s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$
$p_\theta(\tau)$		Probability of trajectory τ under π_θ

EQ5 — Performance Objective

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t, s_{t+1}) \right]$$

Objective: θ* = argmaxθ J(θ) *(Non-convex; high-dimensional; no closed-form gradient — Section 3 shows how to compute ∇θ J(θ) without a dynamics model.)*

From Cost Functions to Reward Functions

In trajectory optimization (Lecture 4), we minimized a cost function ℓ(q, dq/dt, u) over a fixed horizon with known dynamics. The performance objective J(θ) is the mirror image: we maximize a reward over a potentially infinite horizon with unknown dynamics. The critical difference is that trajectory optimization solves for a trajectory, while RL solves for a policy — a function that works across the full distribution of situations the robot will encounter. This is why RL generalizes better to novel conditions.

Exploration, Exploitation, and Policy Types

Exploration vs. Exploitation
Exploiting current knowledge yields the highest expected short-term reward; exploring new actions may reveal better long-run strategies.

ε-greedy strategy:
Prob $(1-\varepsilon)$ → exploit (best-known action)
Prob ε → explore (random action)

Anneal ε from ~1 toward 0 as training converges.

Key Insight: A choice that propagates through every algorithm

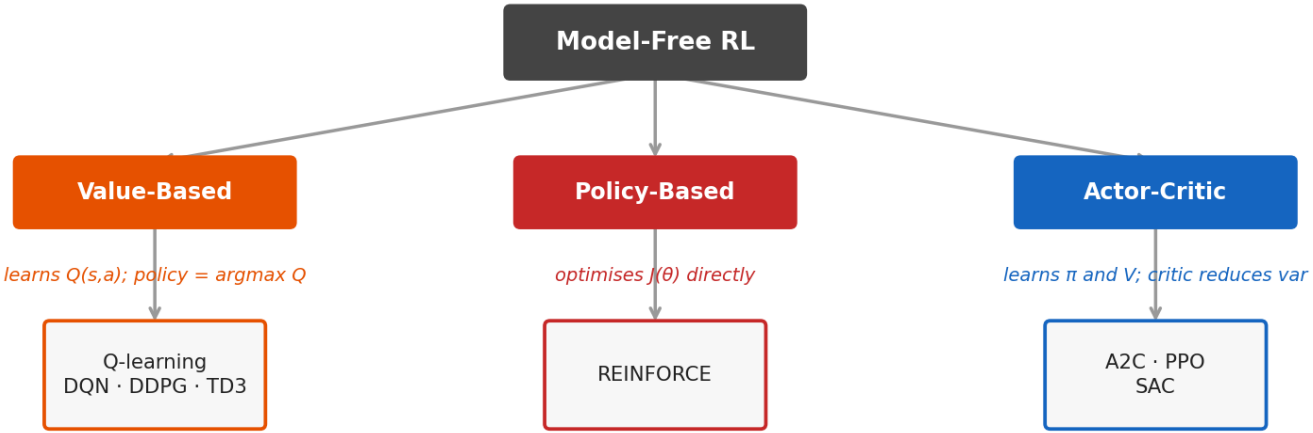
The decision between deterministic and stochastic policies is not cosmetic. It determines whether exploration must be handled externally or is built into the policy itself. It determines whether the policy gradient is estimated through sampled actions or computed analytically.

Deterministic policy: $\mu_\theta : S \to A$ — maps each state to a single action. Used in DDPG, TD3; requires external noise injection (e.g., OU noise) for exploration.

Stochastic policy: $\pi_\theta : S \to \Delta(A)$ — maps each state to a probability distribution. Used in PPO, SAC; randomness serves both exploration and regularization.

Design implication: Value-based methods use deterministic policies with added noise; actor-critic methods use stochastic policies with entropy-based exploration.

The Three Families of Model-Free RL



Value-Based Methods: Learn $Q\pi(s,a)$ or $V\pi(s)$; derive policy as $\text{argmax}_a\, Q(s,a)$. Representatives: Q-learning, DQN, DDPG, TD3.

Policy-Based Methods: Optimize $J(\theta)$ directly by gradient ascent; no value function learned. Representative: REINFORCE.

Actor-Critic Methods: Learn both policy (actor) and value function (critic); actor updates via policy gradient using critic estimates. Representatives: A2C, PPO, SAC.

Value-Based Methods: Q-Learning to TD3

Q-learning (Watkins, 1989)
Tabular Q(s,a) via TD backup; foundational; limited to small discrete spaces.

DQN (Mnih et al., 2015)
Deep network Q-function with experience replay and target networks; discrete actions only.

DDPG (Lillicrap et al., 2016)
Deterministic actor μ_θ(s) extends Q-learning to continuous actions; suffers from Q-overestimation bias.

TD3 (Fujimoto, Hoof, Meger, ICML 2018)
Fixes overestimation via clipped double-Q, delayed policy updates, and target smoothing; continuous-control standard before SAC.

The continuous-control problem
In a discrete space, argmax_a Q(s,a) is a simple lookup. Over a continuous 7-DOF arm, it requires solving an optimization problem at every timestep during both training and deployment.

DDPG and TD3 sidestep this by learning a deterministic actor that parameterizes the argmax. This couples actor and critic so tightly that Q-overestimation corrupts actor updates directly.

Algorithm 1 Q-Learning (Watkins, 1989)

Initialize $Q(s, a) \leftarrow 0$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}$
for each episode **do**
    Observe initial state s_0
    for each timestep t **do**
        With probability ε: select $a_t \leftarrow$ random action
        Otherwise: select $a_t \leftarrow \arg\max_a Q(s_t, a)$
        Execute a_t, observe r_t and s_{t+1}
        Update Q-table via TD backup:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \Big[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \Big]$$

        $s_t \leftarrow s_{t+1}$
    end for
end for

Algorithm 1 Deep Q-Network (DQN)

Initialize Q-network with parameters θ
Initialize target network with parameters $\theta^- \leftarrow \theta$ *// Key idea 1: target network*
Initialize replay buffer $\mathcal{D}$ with capacity N *// Key idea 2: experience replay*
for each episode **do**
    Observe initial state s_0
    for each timestep t **do**
        With probability ε: select $a_t \leftarrow$ random action *// ε-greedy exploration*
        Otherwise: select $a_t \leftarrow \arg\max_a Q(s_t, a; \theta)$
        Execute a_t, observe r_t and s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$ *// Key idea 2*
        Sample random minibatch $\{(s_j, a_j, r_j, s_{j+1})\} \sim \mathcal{D}$ *// Key idea 2*
        Compute TD target using frozen target network: *// Key idea 1*

$$y_j = r_j + \gamma \cdot \max_a Q(s_{j+1}, a; \theta^-)$$

        Minimise loss: $\mathcal{L}(\theta) = \mathbb{E}\Big[(y_j - Q(s_j, a_j; \theta))^2\Big]$
        Update θ by SGD on $\mathcal{L}(\theta)$
        if $t \mod C = 0$ **then**
            $\theta^- \leftarrow \theta$ *// Key idea 1: periodic target network update*
        end if
    end for
end for

Policy-Based Methods and Their Tradeoffs

Pure policy gradient methods optimize J(θ) directly by gradient ascent with no value function. The REINFORCE gradient estimator is unbiased but has very high variance:

EQ6 — REINFORCE Gradient Estimator

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T} G_t^{(i)} \nabla_\theta \log \pi_\theta(a_t^{(i)} \mid s_t^{(i)})$$

where G_t^{(i)} is the discounted return from timestep t in the i-th sampled trajectory.

The variance of this estimator grows with episode length. Robotics tasks have long horizons and sparse rewards — precisely the conditions that maximize variance. In practice, learning fails for these tasks. Actor-critic methods address this by replacing G_t with a lower-variance advantage estimate from a learned critic.

Key Insight: Why pure policy gradients are a stepping stone, not a destination

REINFORCE is theoretically sound but practically unusable for most robotics tasks. The variance of the G_t estimator grows with episode length — and robotics tasks have long horizons and sparse rewards, precisely the conditions that maximize variance. Actor-critic methods solve this by replacing G_t with a lower-variance advantage estimate from a learned critic. Understanding REINFORCE is essential because it is the foundation on which PPO and SAC are built, and because the variance problem it exposes motivates every architectural choice in those algorithms.

Algorithm 1 REINFORCE (Williams, 1992)

Initialize policy network with parameters θ
for each episode **do**
    Collect trajectory $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T)$ by running π_θ
    for each timestep $t = 0, 1, \ldots, T$ **do**
        Compute discounted return:

$$G_t = \sum_{k=0}^{T-t} \gamma^k \, r_{t+k}$$

        Accumulate policy gradient:

$$\nabla_\theta \mathcal{L} \mathrel{+}= G_t \cdot \nabla_\theta \log \pi_\theta(a_t \mid s_t)$$

    end for
    Update parameters: $\theta \leftarrow \theta + \alpha \, \nabla_\theta \mathcal{L}$
end for

Actor-Critic: The Synthesis

Attribute	Value-Based	Policy-Based	Actor-Critic
What is learned	$Q^\pi(s,a)$ or $V^\pi(s)$	Policy π_θ only	Both π_θ and V^π_ϕ
Policy type	Implicit ($\arg\max_a Q$) or deterministic actor	Stochastic	Stochastic (PPO); entropy-regularized (SAC)
Variance	Low (bootstrapped value estimates)	High (full return G_t)	Low (advantage estimate from critic)
Sample efficiency	Moderate to high (TD3/DDPG replay buffer)	Low	Moderate (on-policy PPO) to high (off-policy SAC)
Continuous control	Requires deterministic actor workaround	Natural	Natural
Primary robotics use	TD3 baselines; legacy systems	Rarely used alone	Dominant in all modern robotics RL

Table 1: Comparison of the three model-free RL families across six attributes.

When you read a robotics RL paper, the first question to ask is: which family does this algorithm belong to, and why did the authors choose it? The answer usually depends on three factors — whether a replay buffer is available, whether the action space is continuous, and whether the task allows on-policy data collection at scale. This table gives you the vocabulary to answer that question quickly for any algorithm you encounter.

Algorithm 1 Proximal Policy Optimization (PPO, Schulman et al., 2017)

Initialize policy network π_θ and value network V_ϕ
for each iteration **do**
  Collect N steps from K parallel environments using π_θ
  Store transitions $\{(s_t, a_t, r_t, s_{t+1})\}$ in rollout buffer
  Compute GAE advantage estimates:

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \, \delta_{t+l}, \quad \delta_t = r_t + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$$

whether an action is better than average

  Compute discounted returns $\hat{G}_t$ for value function targets
  Save old policy parameters: $\theta_{\text{old}} \leftarrow \theta$
  for each mini-batch epoch $k = 1, \ldots, K_{\text{epochs}}$ **do**
    Sample random mini-batch from rollout buffer
    Compute probability ratio:

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$$

importance sampling, data from old policy

    Compute clipped surrogate objective *(core PPO innovation)*:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\Big[\min\Big(r_t(\theta)\,\hat{A}_t,\ \text{clip}(r_t(\theta),\, 1{-}\epsilon,\, 1{+}\epsilon)\,\hat{A}_t\Big)\Big]$$

policy update small enough that reuse remains valid

    Compute value loss: $L^{\text{VF}}(\phi) = \mathbb{E}_t\left[\left(V_\phi(s_t) - \hat{G}_t\right)^2\right]$
    Compute total loss:

$$L(\theta, \phi) = -L^{\text{CLIP}}(\theta) + c_1 \, L^{\text{VF}}(\phi) - c_2 \, H(\pi_\theta)$$

    Update θ and ϕ by Adam on $L(\theta, \phi)$
  end for
end for

Model-Based vs. Model-Free: A Fundamental Axis

Model-Free RL

Learns a policy and/or value function from interactions without modelling T(s' | s, a).

Properties:
No model bias; simpler to implement; dominant in robotics practice today.

Representatives:
Q-learning, DQN, TD3, REINFORCE, A2C, PPO, SAC

Model-Based RL

Learns an approximate model of T(s' | s, a) and uses it to plan or generate synthetic training data.

Properties:
Improved sample efficiency; model errors compound over long rollouts; more complex to implement.

Representatives:
Dyna-Q, PlaNet, Dreamer, V-JEPA 2

Key Insight: Why robotics mostly uses model-free RL today

Model-based RL is more sample-efficient in real environment interactions — but for most robotics tasks, simulation provides effectively unlimited interactions at low cost. The sample efficiency advantage of model-based RL is therefore less compelling than the simplicity and robustness of model-free methods when training in parallel simulation. Model-based RL becomes more attractive for tasks where simulator fidelity is insufficient for contact-rich dynamics, and for tasks where a world model has intrinsic value beyond policy training — such as planning and policy evaluation.

The Policy Gradient Theorem

Symbol	Domain	Meaning
∇_θ log π_θ(a_t | s_t)	ℝ^d	Score function (log-derivative of the policy)

EQ7 — Policy Gradient Theorem

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot G_t \right]$$

Derivation sketch (full derivation on next slide):

1. Expand: ∇_θ J(θ) = ∇_θ ∫ p_θ(τ) R(τ) dτ
2. Apply log-derivative identity: ∇_θ p_θ(τ) = p_θ(τ) ∇_θ log p_θ(τ). This rewrites the gradient as an expectation, enabling Monte Carlo estimation.
3. Expand ∇_θ log p_θ(τ). Initial state distribution and transition dynamics do not depend on θ — their gradients are zero and cancel out.
4. Result is an expectation over sampled trajectories, estimable without any knowledge of the environment's dynamics.

Deriving the Policy Gradient: Step by Step

Step 1: Expand the gradient

$$\nabla_\theta J(\theta) = \nabla_\theta \int p_\theta(\tau)\, R(\tau)\, d\tau = \int \nabla_\theta p_\theta(\tau)\, R(\tau)\, d\tau$$

Leibniz rule passes the gradient through the integral under mild regularity conditions.

Step 2: Apply the log-derivative identity

$$= \int p_\theta(\tau)\, \nabla_\theta \log p_\theta(\tau)\, R(\tau)\, d\tau = \mathbb{E}_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau)\, R(\tau)]$$

∇_θ log f = (∇_θ f) / f rewrites the gradient as an expectation, enabling Monte Carlo estimation.

Step 3: Expand the trajectory log-probability

$$\nabla_\theta \log p_\theta(\tau) = \nabla_\theta \log p(s_0) + \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) + \sum_{t=0}^{T} \nabla_\theta \log T(s_{t+1} \mid s_t, a_t)$$

Initial state distribution and transition dynamics do not depend on θ — their gradients vanish and cancel.

Step 4: Final form (policy gradient theorem)

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot G_t \right]$$

Only the policy π_θ remains. Transition dynamics eliminated — the estimator is model-free.

Key Insight: The cancellation that makes model-free RL possible

Step 3 is the pivotal moment. The transition dynamics T(s' | s, a) appear in the log-probability of a trajectory, but because they do not depend on θ, their gradients vanish. What remains is the gradient of the policy alone. This means ∇_θ J(θ) can be estimated from sampled trajectories without any knowledge of environment dynamics.

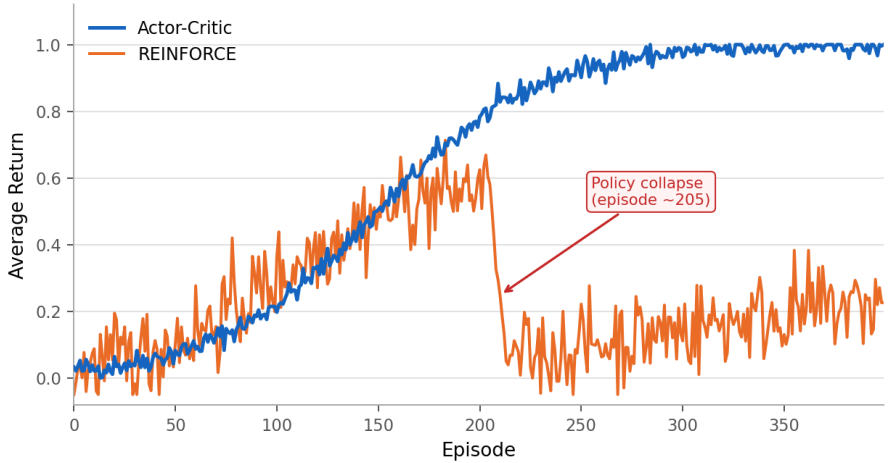
REINFORCE and the Variance Problem

REINFORCE algorithm:

```
Initialize θ
For each episode:
  Collect τ = (s₀,a₀,r₀,...,s_T) under
π_θ
  For each timestep t:
    G_t = Σ_{k=0}^{T-t} γ^k r_{t+k}
    θ ← θ + α ∇θ log π_θ(a_t|s_t) · G_t
```

Why variance is catastrophic:

G_t accumulates all stochastic noise of the entire episode. For sparse binary manipulation reward, most episodes return zero — gradient estimates are noise-dominated. A large gradient step can collapse a partially learned policy irreversibly, requiring millions of new interactions to recover.



Baselines and the Advantage Estimator

EQ8 — Policy Gradient with Baseline

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \cdot (G_t - b(s_t)) \right]$$

Unbiasedness: The expectation of ∇_θ log π_θ(a_t | s_t) · b(s_t) is zero for any b(s_t) that does not depend on a_t, because integrating the score function over the action distribution under π_θ yields zero by definition. Adding a state-dependent baseline therefore reduces variance without introducing any bias.

$$\mathbb{E}_{a_t \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(a_t \mid s_t)] = \int \pi_\theta(a \mid s_t) \nabla_\theta \log \pi_\theta(a \mid s_t) \, da$$

Optimal baseline: b(s_t) = Vπ(s_t), yielding the advantage Aπ(s_t, a_t) = G_t − Vπ(s_t).

- Positive advantage: the chosen action exceeded average expected return.
- Negative advantage: the chosen action fell below average expected return.
- The better the critic estimates Vπ, the lower the variance of the policy gradient signal.

Key Insight: The baseline insight

The score function ∇_θ log π_θ(a_t | s_t) points in the direction that increases the probability of action a_t. Multiplying by the raw return G_t pushes the policy toward actions from successful trajectories, but G_t includes all the noise of the entire episode. Multiplying by the advantage Aπ focuses the update on whether the specific action at time t was responsible for the deviation from average performance. This is the conceptual move from REINFORCE to actor-critic, and it reduces variance by orders of magnitude in practice.

Generalized Advantage Estimation

Symbol	Domain	Meaning
δ_t	$\mathbb{R}$	TD error at time t
$\hat{V}_\phi^\pi(s)$	$\mathbb{R}$	Learned value function estimate (critic)
λ	$[0, 1]$	GAE smoothing parameter
$\hat{A}_t^{\text{GAE}}$	$\mathbb{R}$	GAE advantage estimate

EQ9 — TD Error

$$\delta_t = R_t + \gamma \hat{V}_\phi^\pi(s_{t+1}) - \hat{V}_\phi^\pi(s_t)$$

EQ10 — GAE Advantage Estimate

$$\hat{A}_t^{\text{GAE}} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \, \delta_{t+l}$$

λ parameter interpretation:

- λ = 0: one-step TD estimate — low variance, high bias (trusts the critic completely).
- λ = 1: full Monte Carlo minus value function — low bias, high variance (trusts sampled returns).
- λ = 0.95: the standard in robotics RL; Isaac Lab PPO default. Balances bias and variance empirically well across most tasks.

GAE is the standard advantage estimator in PPO and A2C implementations. If the policy is slow to improve, try increasing λ; if training is high-variance and unstable, reduce it.

The Actor-Critic Architecture

Symbol	Domain	Meaning
L_actor(θ)	$\mathbb{R}$	Actor loss (advantage)
L_critic(φ)	$\mathbb{R}$	Critic loss
c_1, c_2	$\mathbb{R}$_{>0}	Coefficients
H(π_θ)	$\mathbb{R}$	Policy entropy

EQ11 — Actor Loss

$$L_{\text{actor}}(\theta) = -\mathbb{E}_t\left[\hat{A}_t^{\text{GAE}} \cdot \log \pi_\theta(a_t \mid s_t)\right]$$

EQ12 — Critic Loss

$$L_{\text{critic}}(\phi) = \mathbb{E}_t\left[\left(\hat{V}_\phi^\pi(s_t) - G_t\right)^2\right]$$

EQ13 — Combined A2C Loss

$$L(\theta, \phi) = L_{\text{actor}} + c_1 \cdot L_{\text{critic}} - c_2 \cdot H(\pi_\theta)$$

The $-c_2 \cdot H(\pi\_\theta)$ term encourages high entropy and exploration — the direct precursor to SAC's entropy-maximization objective.

Algorithm 1 Actor-Critic (A2C)

Initialize actor network π_θ and critic network $\hat{V}_\phi^\pi$
for each iteration **do**
  Collect trajectory $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T)$ using π_θ
  for each timestep $t = 0, 1, \ldots, T$ **do**
    Compute TD error: *// critic evaluates the action*

$$\delta_t = r_t + \gamma \hat{V}_\phi^\pi(s_{t+1}) - \hat{V}_\phi^\pi(s_t)$$

    Compute GAE advantage estimate: *// variance reduction*

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \, \delta_{t+l}$$

  end for
  Compute actor loss: *// policy improvement*

$$L_{\text{actor}}(\theta) = -\mathbb{E}_t\left[\hat{A}_t \cdot \log \pi_\theta(a_t \mid s_t)\right]$$

  Compute critic loss: *// value function improvement*

$$L_{\text{critic}}(\phi) = \mathbb{E}_t\left[\left(\hat{V}_\phi^\pi(s_t) - G_t\right)^2\right]$$

  Compute combined loss:

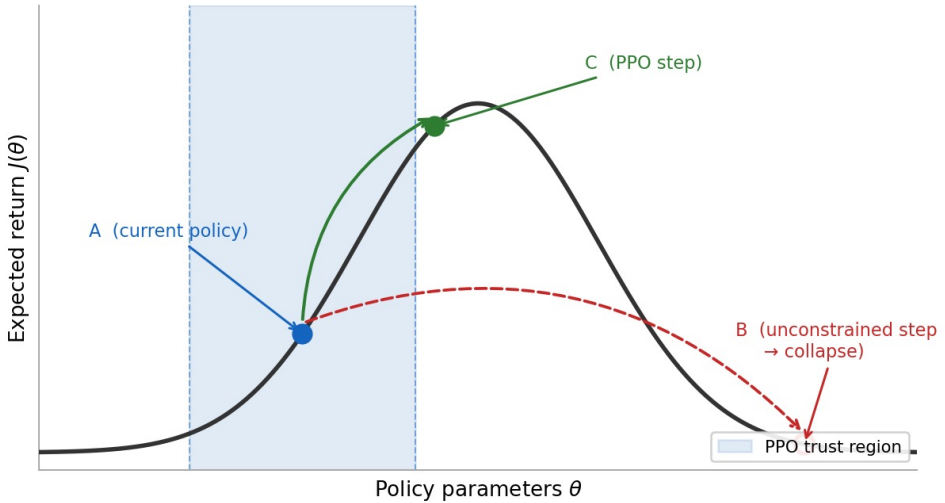
$$L(\theta, \phi) = L_{\text{actor}} + c_1 \, L_{\text{critic}} - c_2 \, H(\pi_\theta)$$

  Update θ and ϕ by Adam on $L(\theta, \phi)$
end for

The Problem PPO Solves

The policy collapse problem

The actor-critic update imposes no constraint on step size. A large gradient step violates the on-policy assumption: advantage estimates become invalid under the new policy, and performance can collapse irreversibly. Recovery requires millions of new environment interactions.



Two solutions:

- TRPO — hard trust-region constraint via natural gradient optimization. Principled but computationally expensive per update.

- PPO — soft constraint via probability-ratio clipping. Nearly as effective at a fraction of the cost. All practical robotics RL uses PPO or SAC rather than TRPO.

PPO's key intuition: Clip r_t(θ) = π_θ/π_old so the objective never rewards moving it too far from 1. The policy can improve, but not catastrophically.

PPO: The Clipped Surrogate Objective

Symbol	Domain	Meaning
θ_old	ℝ^d	Policy parameters at data-collection time
r_t(θ)	ℝ_{>0}	Probability ratio: new policy / old policy
ε	(0, 0.3)	Clipping hyperparameter
A-hat_t	ℝ	GAE advantage estimate

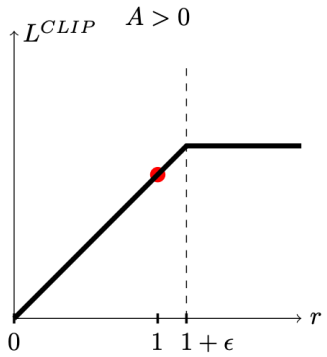
EQ14 — Probability Ratio

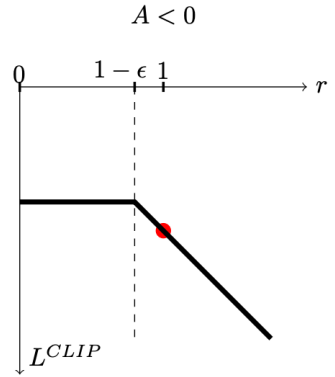
$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

EQ15 — PPO Clipped Surrogate Objective

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

Clipping: If A-hat_t > 0, increase π_θ(a_t|s_t) only up to (1+ε)×old probability. If A-hat_t < 0, penalize only down to (1-ε) ×old probability. The min ensures a lower bound on the unclipped objective.





SAC: Maximum Entropy RL

Symbol	Domain	Meaning
α	ℝ_{>0}	Temperature parameter (weight on entropy)
H(π_θ(·|s_t))	ℝ	Entropy of the policy's action distribution
D		Replay buffer

EQ16 — SAC Entropy-Augmented Objective

$$J_{\mathrm{SAC}}(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T} \gamma^t \left(R(s_t, a_t, s_{t+1}) + \alpha \, H(\pi_\theta(\cdot \mid s_t)) \right) \right]$$

Off-policy: Transitions from any prior policy stored in D and reused, improving sample efficiency.
Reparameterization: Handles continuous actions without inner argmax (name only; no derivation required).
Auto temperature tuning: Haarnoja et al. (arXiv:1812.05905) adjusts α to maintain target entropy, removing the most sensitive hyperparameter.

Key Insight: SAC's entropy bonus is not just about exploration

The entropy term does double duty. It encourages exploration, yes — but more importantly, it regularizes the policy against becoming brittle. A policy that assigns near-unit probability to a single action in each state generalizes poorly to small perturbations in dynamics. The entropy bonus keeps the policy soft, maintaining a distribution of behaviors. This is precisely what makes SAC policies more robust.

PPO vs. SAC: A Framework for Choosing

Attribute	PPO	SAC
Policy class	On-policy	Off-policy
Sample efficiency (env. steps)	Lower — requires fresh rollouts each update	Higher — replay buffer D reuses past data
Wall-clock efficiency	Higher with GPU parallelism (4096+ envs)	Higher when env. throughput is limited
Stability	Clipping; robust to hyperparameter choices	Entropy + auto temperature; stable across reward scales
Primary robotics use	Locomotion with GPU parallel simulation	Contact-rich manipulation on hardware or small-scale sim
2024–2025 results	ANYmal, Unitree G1/H1, humanoid locomotion	SERL, HIL-SERL, dexterous manipulation

Key decision variable: *If thousands of parallel environments are available, PPO's large on-policy batches dominate. If each environment step carries high cost, SAC's replay buffer makes each interaction more valuable.*

PPO in Practice: Implementation Details

1. **Rollout collection.** Collect N steps from K parallel environments — N×K transitions per update. Standard: K=4096, N=24 in Isaac Lab (supports 4096 envs on a single GPU; locomotion training compresses from days to hours).
2. **Advantage normalization.** Normalize A-hat_t to zero mean and unit variance within each mini-batch. Stabilizes learning across reward scales without changing the optimal policy.
3. **Value function clipping.** Clip the critic update symmetrically around the old value estimate. Reduces variance.
4. **Entropy bonus coefficient.** Begin with c_2=0.01 for locomotion. Reduce if policy stays excessively stochastic; increase if it collapses to near-determinism early.
5. **Learning rates.** Actor and critic typically share 3×10^{-4} with the Adam optimizer.
6. **Episode termination masking.** Isaac Lab resets environments individually on fall or timeout. Terminal transitions must be masked to prevent bootstrapping through episode boundaries.

Learning Connection: PPO and the Impedance Controller from Lecture 3

When training a locomotion policy with PPO in Isaac Lab, the policy is a small MLP that maps proprioceptive state to joint position targets. Those targets are passed to a low-level PD controller — exactly the position controller from Lecture 3. PPO is not replacing the classical controller; it is learning the high-level behavior that tells it what to track. This architecture underlies every major legged locomotion result from ANYmal and Unitree H1 to the most recent humanoid systems. The classical controller handles stable low-level tracking; RL handles high-level behavioral adaptation.

A Worked Comparison: PPO and SAC on the Same Task

Task: block-stacking on a 7-DOF arm (MuJoCo)

PPO Formulation

Environment: 512 parallel MuJoCo envs. Rollout N=48 steps.
Obs/Action: Joint pos., vel., EE pose, obj. pose / $\Delta q\_des \in \mathbb{R}^7$.
Reward: $-0.1\|p\_ee - p\_target\| + 1.0 \cdot 1[\text{success}]$.
Key hyperparameters: $\varepsilon=0.2$, $\gamma=0.99$, $\lambda=0.95$, 10 mini-batch epochs/rollout, $c_2=0.01$.
Convergence: ~50M environment steps. Policy: 3-layer MLP, 256 units/layer.

SAC Formulation

Environment: Single MuJoCo env. Replay buffer: 10^6 transitions.
Obs/Action: Identical to PPO.
Reward: Same formulation; SAC robust to dense reward scaling via auto temperature tuning.
Key hyperparameters: α auto-tuned (target entropy $-|\alpha|$), batch size 256, clipped double-Q.
Convergence: ~500K environment steps — far fewer than PPO, but on a single environment.

Comparison: PPO requires orders of magnitude more environment interactions but parallelizes efficiently across GPUs. SAC uses far fewer interactions but cannot trivially exploit GPU parallelism in the same way — off-policy replay breaks the single-pass structure of on-policy rollout collection.

Beyond PPO and SAC: The Broader Landscape

Algorithm	Family	On/Off	Key Advantage	Key Limitation	Representative Use
PPO	Actor-Critic	On-policy	Stable; GPU-parallel; robust	Sample-hungry in env. steps	Locomotion; all parallelisable tasks
SAC	Actor-Critic	Off-policy	Sample-efficient per step	Less natural GPU parallelism	Contact-rich manipulation; hardware RL
TD3	Value-Based	Off-policy	Simple; deterministic policy	Brittle to Q-overestimation	Manipulation baselines; legacy systems
Dreamer	Model-Based	Off-policy	Trains on imagined rollouts	Model error accumulates; complex	Vision-based tasks

Key Insight: Choosing an algorithm is choosing a set of assumptions

Every algorithm in this table embeds assumptions about the training environment: how many interactions are available, whether the environment is parallelisable, whether a world model is feasible, and whether the task requires a stochastic or deterministic policy. There is no universally best algorithm. The skill developed in this lecture is the ability to match the algorithm's assumptions to the task's constraints. For the majority of robotics tasks you will encounter in this course and in the literature, PPO or SAC will be the appropriate choice — but knowing why requires understanding the full landscape this table describes.

The Reward Function: Formal Theory

Symbol	Domain	Meaning
R'(s,a,s')	$\mathbb{R}$	Shaped reward function
Φ(s)	$\mathbb{R}$	Potential function (state-dependent scalar)
F(s,a,s')	$\mathbb{R}$	Shaping term

EQ17 — Potential-Based Reward Shaping

$$R'(s,a,s') = R(s,a,s') + F(s,a,s'), \quad \text{where} \quad F(s,a,s') = \gamma\Phi(s') - \Phi(s)$$

Theorem (Ng, Harada, Russell, ICML 1999): Any agent achieving the optimal policy under R' also achieves the optimal policy under R, provided F is potential-based as defined above. Any non-potential-based additive shaping term can change the set of optimal policies.

A sparse binary reward $R \in \{0,1\}$stalls training — the agent rarely discovers success through random exploration. Define. $\Phi(s) = -\|p_{\text{ee}} - p_{\text{block}}\|$ and add the shaping term $F = \gamma\Phi(s') - \Phi(s)$. The agent now receives a dense signal guiding it toward contact. The shaping term accelerates discovery without redefining the goal.

The Reward Spectrum: Sparse, Dense, and Shaped

Property	Sparse	Dense	Potential-Based Shaped
Definition	$R \in \{0,1\}$ at task completion	$R = -\|x - x^*\|$ or similar	$R' = R + \gamma\Phi(s') - \Phi(s)$
Pros	Clean; optimal policy well-defined	Easy to learn from; dense gradient signal	Preserves optimal policy; reduces credit assignment difficulty
Cons	Exploration failure in large state spaces	Reward hacking; policy can maximise metric without completing task	Requires well-designed Φ; poor Φ can slow learning
Typical robotics use	Final fine-tuning; short-horizon tasks	Velocity tracking; reaching tasks	Structured manipulation with known intermediate states

Specification question: *The most common failure mode in robotics RL is a dense reward that the policy optimizes in an unintended way. Always ask: does the reward measure task completion, or a proxy that correlates during training but can be decoupled by a sufficiently capable optimizer?*

Reward Hacking: When the Metric Becomes the Target

Example 1 — Locomotion height reward. A policy rewarded for maintaining a high pelvis height learns to stand on one foot with the other leg extended horizontally. Fixed by replacing with velocity tracking plus energy penalty.

Example 2 — Manipulation proximity reward. A policy rewarded for minimizing distance between the end-effector and the target object learns to swipe the object using the end-effector rather than grasping it. The most common failure mode in early manipulation RL experiments.

Example 3 — Survival bonus exploitation. A locomotion policy rewarded for staying alive plus velocity tracking learns to move as slowly as possible, barely satisfying the velocity constraint while maximising episode length.

Key Insight: Goodhart's Law in RL

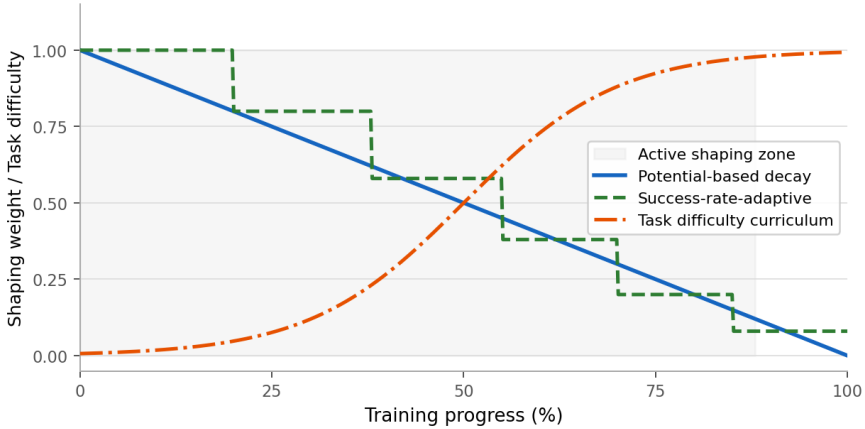
There is a well-known principle in economics: when a measure becomes a target, it ceases to be a good measure. RL makes this operational with brutal efficiency. Given millions of training steps and thousands of simultaneous exploration agents, any gap between the reward function and the true objective will be found and exploited. Designing a reward that captures exactly what you want — and nothing more — is often the hardest problem in a robotics RL project. This is a specification problem, and it requires the same rigor as writing a formal proof.

Reward Curriculum: Shaping Over Training

Potential-based initialization. Begin with dense shaped reward; progressively reduce shaping as competence grows, ending with the sparse task-completion reward.

Success-rate-adaptive shaping. Adjust shaping weight with rolling success rate: full shaping below 10% success; reduce by fixed factor above 60%. Removes manual scheduling.

Task difficulty curriculum. Isaac Lab terrain curriculum: flat → rough terrain → stairs → obstacles as performance improves.



Key Insight: Curriculum design is reward engineering over time

A static reward function makes a single bet: that the agent can discover good behaviors from the initial reward signal through exploration alone. A curriculum hedges that bet by changing the learning problem as the agent's capability grows. Shaping adds structure in the reward space; curriculum adds structure in the task space. Both are forms of prior knowledge injection, and both require the same rigor in design to avoid creating new pathologies.

Intrinsic Motivation and Reward-Free Exploration

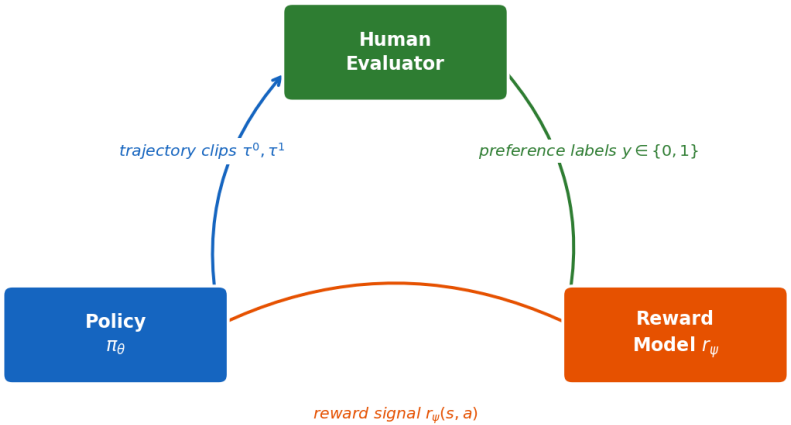
Curiosity-driven exploration (Pathak et al., ICML 2017)
Intrinsic reward = forward dynamics prediction error: r_int = ‖Ţ(s_{t+1}|s_t,a_t) − s_{t+1}‖². Rewards visiting states the model cannot predict. Effective for sparse-reward manipulation where the agent must discover grasps before external reward appears.

Random Network Distillation / RND (Burda et al., ICLR 2019)
Trains a predictor network to match a fixed random network. Novel states have high prediction error; visited states have low error. Simpler than curiosity and more robust to the “noisy TV” pathology (high error due to stochasticity).

When to use intrinsic motivation:
A tool for the exploration phase, not a replacement for task reward. Unnecessary for most tasks with dense reward or teacher-student supervision.

Essential for very sparse rewards with large state spaces — open-ended tool use or multi-stage assembly with no intermediate milestones.

Beyond Hand-Crafted Rewards: Preference Learning



Bradley-Terry preference model:
$P(\tau^1 \succ \tau^0) = \sigma(r\_\psi(\tau^1) - r\_\psi(\tau^0))$

Robotics relevance: Enables reward specification for tasks easy to recognise but hard to specify analytically: folding laundry, furniture assembly, dexterous manipulation of deformable objects. The human need only compare behaviors pairwise.

Limitation: Requires human in the loop; bottlenecked by the number of pairwise labels available during long training runs.

Key Insight: Inverting the reward design problem
Standard reward engineering asks: “What numerical function should I optimize?” Preference learning asks: “Which of these two behaviors do I prefer?” The second is far easier for humans to answer for tasks with complex, multi-attribute success criteria. The price is that the reward is now a learned model — a slightly wrong reward model will produce reward hacking against the model rather than the hand-crafted function. This transforms reward engineering from a specification problem into a model-accuracy problem.

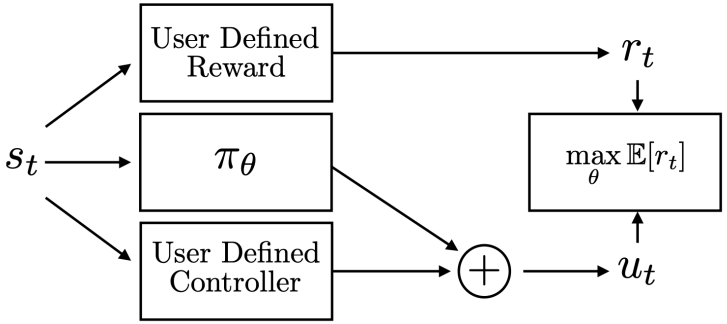
Foundation Model Rewards and the Emerging Frontier

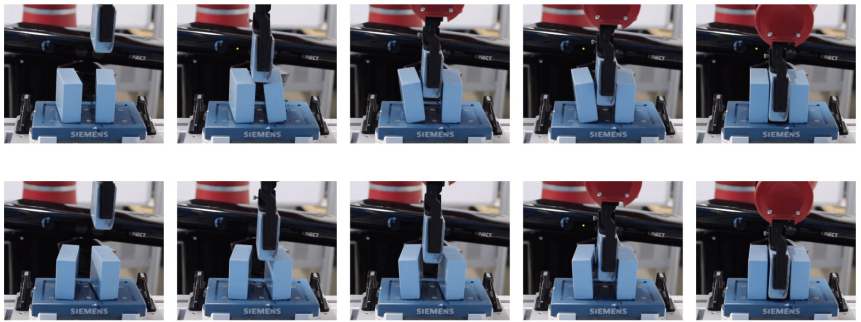
Method 1 — Learned success classifiers (HIL-SERL, Luo et al., Science Robotics 2024)
Binary classifier trained from human-labeled success/failure demonstrations; success probability serves as reward signal. Demonstrated on cable routing and connector insertion — super-human performance after hours of real-robot training. Key advantage: no analytical reward specification.

Method 2 — VLM-based reward signals
CLIP, GPT-4V are prompted to evaluate robot behaviors and generate scalar rewards; reward specified entirely in natural language. Current limitations: compute cost per query, prompt sensitivity, difficulty with fine-grained contact-level judgments.

Key Insight: The direction reward engineering is moving
The progression hand-crafted → potential-based → preference-based → classifier-based → VLM-based rewards represents a steady relaxation of the specification burden on the designer. Each step reduces the precision of knowledge required to define the reward, at the cost of introducing new sources of approximation error. The field has not yet converged on a dominant paradigm — hand-crafted rewards remain standard for locomotion, while learned rewards are gaining ground for manipulation tasks where success is visually recognisable but analytically underspecified.

Residual Policy Architecture





Symbol	Domain	Meaning
π_base(s)	A	Fixed classical controller output
π_θ(s)	A	Learned residual policy output
u	A	Total control action

EQ18 — Residual Control Law

$$u = \pi_{\text{base}}(s) + \pi_\theta(s)$$

Result (Johannink et al., ICRA 2019)
Block insertion on a real Sawyer arm — task learned within three hours of real-robot training. Without the base controller, the same task could not be solved within the same wall-clock budget.

Why Residual RL Works

Reduced exploration burden. The residual policy starts near a reasonable behavior rather than at random. The policy gradient has a shorter path to a good solution because the base controller already handles the gross dynamics correctly.

Safety during training. The base controller ensures a reasonable gross trajectory even when the residual is random, preventing hardware-damaging movements during early training — critical for industrial assembly where contact approaches must be precise.

Interpretability. Classical and learned components are separate and independently inspectable. Engineers can reason about base behavior in isolation when the residual produces unexpected corrections.

2024–2025 extension: MPC-residual architectures for legged locomotion combine an MPC trajectory planner as the base with an RL residual for terrain adaptation, achieving 20–30% improvement in asymptotic reward over pure RL with zero-shot generalization to new terrain types.

> The impedance controller from Lecture 3 can serve directly as the base policy in a residual RL system. The spring-damper model provides compliant, safe contact behavior; the RL policy adds only the learned correction for contact dynamics the spring-damper model cannot handle alone.

Summary

Act 1 — Why learning: Classical control fails at scale when accurate contact models are unavailable; RL is the appropriate tool for tasks requiring adaptation to distributions of conditions that cannot be enumerated in advance.

Act 2 — The formalism: The MDP provides the mathematical language: five components, Bellman equations, and the performance objective as the explicit optimization target.

Act 3 — The landscape: The three families of model-free RL differ in what they learn and how they explore; actor-critic methods dominate robotics practice.

Act 4 — The algorithms: The policy gradient theorem gives a computable, model-free gradient; GAE and the actor-critic architecture reduce variance; PPO and SAC are the two practical implementations with well-understood on/off-policy tradeoffs.

Act 5 — The engineering: Reward engineering — from potential-based shaping through hacking, curriculum, and learned reward functions — is the dominant open problem in applied robotics RL, requiring the same rigor as algorithm design.

The Limits of RL

Reward specification. For tasks where success is recognisable but analytically underspecified — folding laundry, dexterous assembly with complex geometry, contact-rich insertion with variable tolerances — writing a reward function that produces the desired behavior is extremely difficult or impossible.

Sample efficiency at task complexity. Even with thousands of parallel environments, tasks with very long horizons and sparse intermediate rewards frequently fail to converge within practical training budgets. A human demonstrator provides the dense supervisory signal needed to guide the policy through intermediate states.

If a human can show you how to do the task, why run 200 million RL steps to discover the same behavior?