

Robot Learning with `lerobot`: A Tutorial

Francesco Capuano  ... Adil Zouitine  Pepijn Kooijmans  Thomas Wolf  Michel Aractingi 

 École Normale Supérieure Paris-Saclay,  Hugging Face

Abstract



Hugging Face

Recent advancements in foundation models have enabled unprecedented progress in robotics, particularly in the development of increasingly more capable generalist robotics models. Leveraging modern learning-based approaches, the robotics community produced the first general models for tasks including locomotion and manipulation, and many said results have been openly released to the open-source community. The premise of generalist models is to learn control strategies across tasks and different robot embodiments, a long-standing challenge in robot learning. This work aims at being a tutorial on the most common techniques used for modern robot learning, covering both the conceptual underpinnings of the most prominent approaches, and pairing them with reproducible implementations using `lerobot` (<https://github.com/huggingface/lerobot>), the full-stack open-source robotics library developed by Hugging Face. This tutorial is designed to serve as a self-contained, structured reference for robotics practitioners working with `lerobot`, as well as a practical guide for researchers in the field of robot learning.

1 Introduction

Autonomous robotics holds the premise of relieving humans from repetitive, tiring or dangerous manual tasks. Consequently, the field of robotics has been widely studied since its first inception in the 1950s. Lately, advancements in Machine Learning (ML) have sparked the development of a relatively new class of methods used to tackle robotics problems, leveraging large amounts of computation and data rather than human expertise and modeling to develop autonomous systems.

Robotics in 2025 sits is increasingly moving away from classical model-based control paradigm, embracing the advancements made in ML, thus unlocking (1) monolithic perception-to-action action pipelines and (2) multi-modal data-driven feature extraction strategies, together with (3) reduced reliance on precise models of the world and (4) a better positioning to benefit from the growing availability of robotics data openly available. While central problems in manipulation, locomotion and whole-body control demand knowledge of rigid-body dynamics, contact modeling, planning under uncertainty, recent results seem to indicate learning can prove as effective as explicit modeling, sparking interest in the field of *robot learning*. This interest can be largely justified considering the significant challenges related to deriving accurate models of robot-environment interactions. Also, because end-to-end learning on large and increasing amounts of data produced *foundation models* capable to semantically reason over multiple input modalities

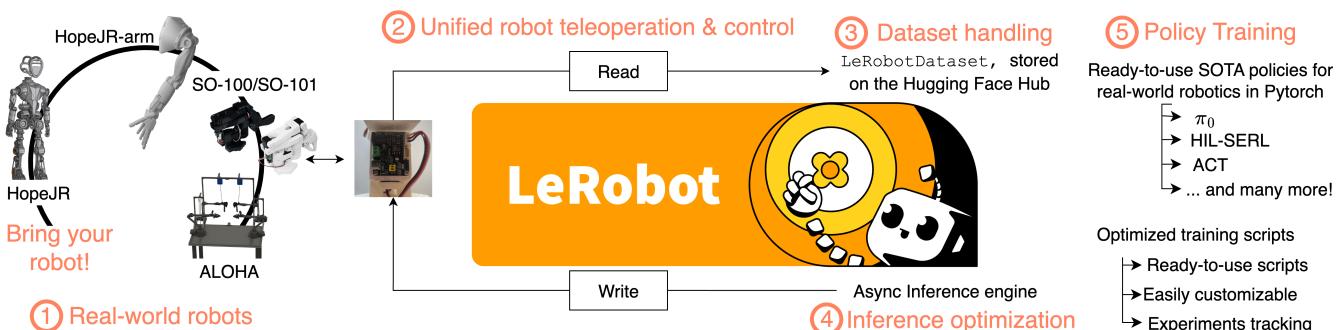


Figure 1 | `lerobot` is an open-source library for end-to-end robotics, supporting SOTA robot learning methods in Pytorch

(images, text, audio, etc.), deriving methods for robotics based on learning seems particularly promising, given the current trends related to the growth of openly available datasets.

At its core, robotics is also an inherently multi-disciplinary field, to the very least considering the diverse skills required in terms of *software* and *hardware*. Integrating learning-based techniques increases the heterogeneity of skills needed to tackle robotics, whether in applications or research. `1erobot` is an open-source library end-to-end integrated with the entire robotics stack. With a strong focus on accessible, real-world robots, (1) `1erobot` supports many, openly available, robotic platforms for manipulation, locomotion and even whole-body control. `1erobot`'s (2) unified, low-level approach to reading/writing robot configurations also allows to extend support for other robots relatively easily. The library also supports (3) a native robotics dataset's format—`LeRobotDataset`—that is used to efficiently share datasets. `1erobot` also supports many state-of-the-art (SOTA) algorithms in robot learning—Reinforcement Learning (RL) and Behavioral Cloning (BC)—with efficient implementations in Pytorch and extended support to experimentation and experiments tracking. Lastly, `1erobot` defines a custom, optimized inference stack for robotic policies decoupling action planning from action execution, proving effective in guaranteeing more adaptability at runtime.

This tutorial serves the double purpose of providing useful references for the science and practical use of common robot learning techniques. To this aim, we strike to provide rigorous yet concise overviews of the core concepts behind the techniques presented, paired with practical examples of how to use these in applications, for practitioners and researchers in the field of robot learning. This tutorial is structured as follows:

- Section 2.1 reviews classical robotics foundations, introducing the limitations of dynamics-based approaches to robotics.
- Section 3 elaborates on the limitations of dynamics-based methods, and introduces learning-based techniques, their upsides and potential limitations.
- Section ?? further describes robot learning techniques that aim at solving single tasks learning from specific expert demonstrations.
- Section ?? presents recent contributions towards generalist models for robotics applications learning from large corpora of multi-robot multi-task data.
- Lastly, Section ?? covers emerging directions in robot learning research, introducing recent works in post-training techniques for robotics foundation models.

Lastly, we complement our presentation of the most common and recent approaches in robot learning with practical code implementations using `1erobot`.

2 Classical Robotics

Know your enemy [...]

Sun Tzu

TL;DR

Learning-based approaches to robotics are motivated by the need to (1) generalize across tasks and embodiments (2) reduce dependency on human expertise (3) leverage historical trends on the production of data—all traditionally overlooked by dynamics-based techniques.

2.1 Explicit and Implicit Models

Robotics is concerned with producing artificial motion in the physical world in useful, reliable and safe fashion. Thus, robotics is an inherently multidisciplinary domain: producing autonomous motion in the physical world requires, to the very least, interfacing different software (motion planners) and hardware (motion executioners) components. Further, knowledge of mechanical, electrical, and software engineering, as well as rigid-body mechanics and control theory have therefore proven quintessential in robotics since the field first developed in the 1950s. More recently, Machine Learning (ML) has also proved effective in robotics, complementing these more traditional disciplines (Connell and Mahadevan, 1993). As a direct consequence of its multi-disciplinary nature, robotics has developed as a rather wide

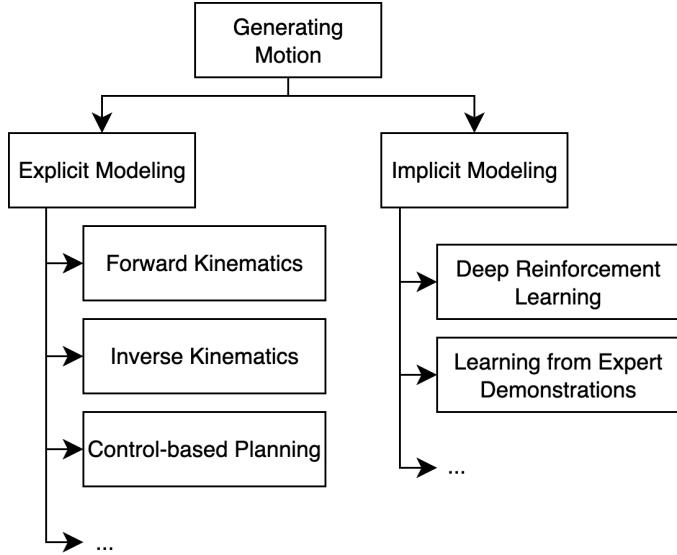


Figure 2 | Overview of methods to generate motion (clearly non-exhaustive, see Bekris et al. (2024)). The different methods can be grouped based on whether they explicitly (*dynamics-based*) or implicitly (*learning-based*) model robot-environment interactions.

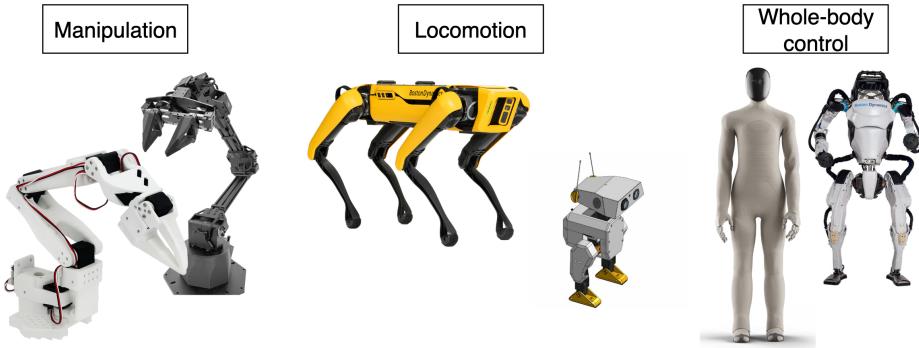


Figure 3 | Different kinds of motions are achieved with potentially very different robotic platforms. From left to right, top to bottom: ViperX, SO-100, Boston Dynamics' Spot, Open-Duck, 1X's NEO, Boston Dynamics' Atlas. This is an example list of robotic platforms and is (very) far from being exhaustive.

array of methods, all concerned with the main purpose of [producing artificial motion in the physical world](#).

Methods to produce robotics motion range from traditional *explicit* models—[dynamics-based](#) methods, leveraging precise descriptions of the mechanics of robots' rigid bodies and their interactions with eventual obstacles in the environment—to *implicit* models—[learning-based](#) methods, treating artificial motion as a statistical pattern to learn given multiple sensorimotor readings (Agrawal; Bekris et al., 2024). A variety of methods have been developed between these two extrema. For instance, Hansen et al. (2022) show how learning-based systems can benefit from information on the physics of problems, complementing a traditional learning method such as Temporal Difference (TD)-learning Sutton and Barto (2018) with Model-Predictive Control (MPC). Conversely, as explicit models may be relying on assumptions proving overly simplistic—or even unrealistic—in practice, learning can prove effective to improve modeling of complex phenomena or complement perception (McCormac et al., 2016). Such examples aim at demonstrating the richness of approaches to robotics, and Figure 2 graphically illustrates some of the most relevant techniques. Such a list is clearly far from being exhaustive, and we refer to Bekris et al. (2024) for a more comprehensive overview of both general and application-specific methods for motion generation. In this section, we wish to introduce the inherent benefits of [learning-based approaches to robotics](#)—the core focus on this tutorial.

2.2 Different Types of Motion

At its very core, robotics deals with producing motion via actuating joints connecting nearly entirely-rigid links. A key distinction between focus areas in robotics is based on whether the generated motion modifies (1) the relative state of the robot with respect to its environment, (2) the absolute state of the environment or (3) a combination of the two (Figure 3).

For instance, (1) may consist in changes in the robot's physical location within its environment. Generally, modifications to a robot's location within its environment may be considered instances of the general *locomotion* problem, further specified as *wheeled* or *legged* locomotion based on whenever a robot makes use of wheels or leg(s) to move in the environment. Effects such as (2) are typically achieved *through* the robot, i.e. generating motion to perform an action inducing a desirable modification, effectively *manipulating* the environment (manipulation). Lastly, an increased level of dynamism in the robot-environment interactions can be obtained combining (1) and (2), thus designing systems capable to move within *and* interact with their environment. This category of problems is typically termed *whole-body control*, and is characterized by a typically much larger set of control variables compared to either locomotion or manipulation alone.

The traditional body of work developed since the very inception of robotics is increasingly more complemented by learning-based approaches. ML has indeed proven particularly transformative across the entire robotics stack, first empowering planning-based techniques with improved state estimation used for traditional planning (Tang et al., 2023) and then end-to-end replacing controllers, effectively yielding perception-to-action methods (Kober et al.). Work in producing robots capable of navigating a diverse set of terrains demonstrated the premise of both dynamics and learning-based approaches for locomotion (Griffin et al., 2017; Ji et al., 2023; Lee et al., 2020; Margolis et al., 2022), and recent works on whole-body control indicated the premise of learning-based approaches to generate rich motion on complex robots, including humanoids (Zhang et al., 2024; ?). Manipulation has also been widely studied, particularly considering its relevance for many impactful applications ranging from high-risk applications for humans (Fujita et al., 2020; Alizadeh and Zhu, 2024; Fujita et al., 2020) to manufacturing (Sanneman et al., 2020). While explicit models have proven fundamental in achieving important milestones towards the development of modern robotics, recent works leveraging implicit models proved particularly promising in surpassing scalability and practical applicability challenges via learning (Kober et al.).

2.3 Example: Planar Manipulation

Robot manipulators typically consist of a series of links and joints, articulated in a chain finally connected to an *end-effector*. Links and joints are considered responsible for generating motion, while the end effector is instead used to perform specific actions at the target location (e.g., grasping/releasing objects via closing/opening a gripper end-effector, using a specialized tool like a screwdriver, etc.).

Recently, the development of low-cost manipulators like the Aloha (Zhao et al., 2023) Aloha-2 (Aldaco et al.) and SO-100/SO-101 (Knight et al.) platforms significantly lowered the barrier to entry to robotics, considering the increased accessibility of these robots compared to more traditional platforms like the Franka Emika Panda arm (Figure 4, (Knight et al.)).

Deriving an intuition as per why learning-based approaches are gaining popularity in the robotics community requires briefly analyzing traditional approaches for manipulation, leveraging tools like forward and inverse kinematics (FK, IK) and control theory. Providing a detailed overview of these methods falls (well) out of the scope of this tutorial, and we refer the reader to works including Siciliano and Khatib (2016); Lynch and Park (2017); Tedrake (a,b) for a much more comprehensive description of these techniques. Here, we mostly wish to highlight what we believe are the main reasons why the field of robotics is increasingly moving *away* from these legacy techniques.

Consider the (simple) case where a SO-100 is restrained from actuating (1) the shoulder pane and (2) the wrist flex and roll motors. This effectively reduces the degrees of freedom of the SO-100 from the original 5+1 (5 actuators + 1 end-effector) to 2+1 (shoulder lift, elbow flex + gripper). As the end-effector does not impact motion in this model, the SO-100 is effectively reduced to the planar manipulator robot presented in Figure 5, where spheres represent actuators, and solid lines indicate links from the base of the SO-100 to the end-effector (*ee*).

Further, let us make the simplifying assumption that actuators can produce rotations up to 2π radians. In practice, this is seldom the case due to movement obstructions caused by the robot body itself (for instance, the shoulder lift cannot produce counter-clockwise movement due to the presence of the robot's base used to secure the SO-100 to its support and host the robot bus), but we will introduce movement obstruction at a later stage.

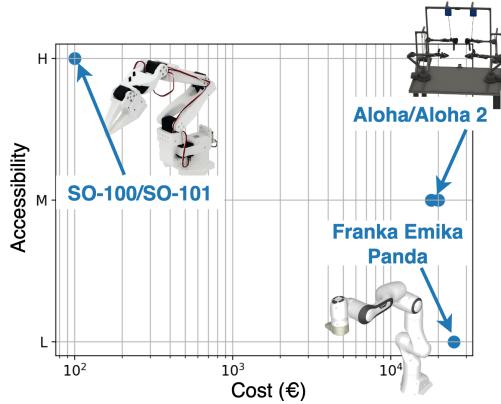


Figure 4 | Cheaper, more accessible robots are starting to rival traditional platforms like the Panda arm platforms in adoption in resource-constrained scenarios. The SO-100, in particular, has a cost in the 100s of Euros, and can be entirely 3D-printed in hours, while the industrially-manufactured Panda arm costs tens of thousands of Euros and is not openly available.

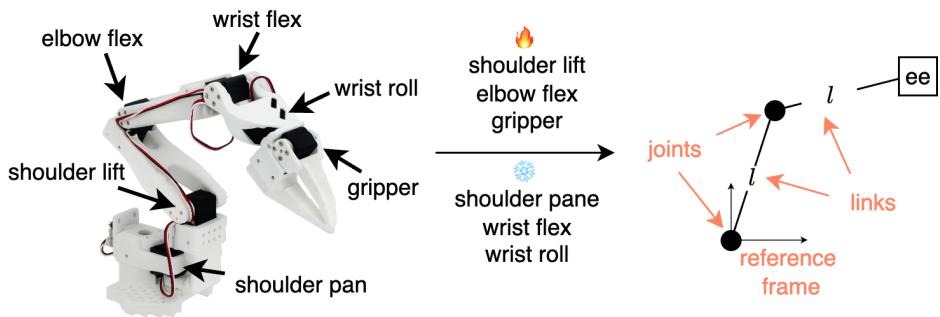


Figure 5 | The SO-100 arm is a 6-dof manipulator arm. Preventing some of its joints (shoulder pane, wrist flex and wrist roll) from actuating, it can be represented as a traditional 2-dof planar manipulator (the gripper joint in the end-effector is not considered towards the count of the degrees of freedom used to produce motion).

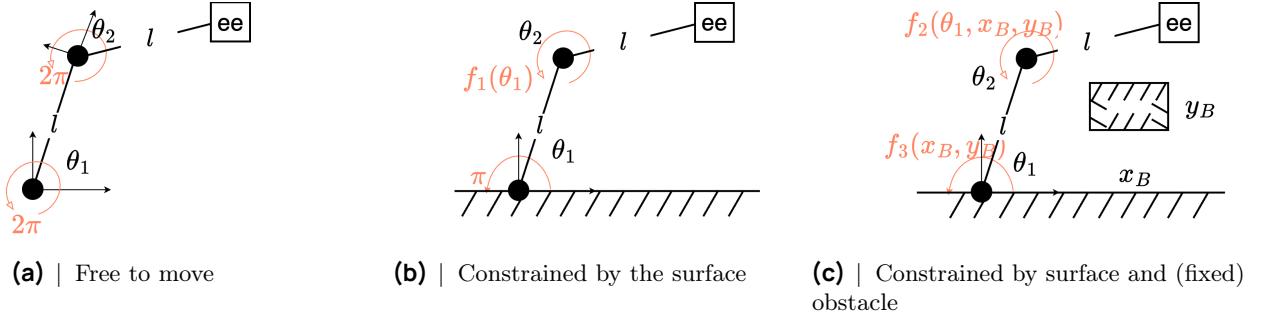


Figure 6 | Planar, 2-dof schematic representation of the SO-100 manipulator under diverse deployment settings. From left to right: completely free of moving ($\mathcal{Q} = [-\frac{\pi}{2}, +\frac{\pi}{2}]^2$); constrained by the presence of the surface; constrained by the surface and presence of obstacles.

All these simplifying assumptions leave us with the planar manipulator of Figure 6a, free of moving its end-effector by controlling the angles θ_1 and θ_2 , jointly referred to as the robot’s *configuration*, and indicated with $q = [\theta_1, \theta_2] \in [\frac{\pi}{2}, +\frac{\pi}{2}]^2$. In this tutorial, we do not cover spatial algebra and frames of reference, and we instead refer the interested reader to Lynch and Park (2017, Chapter 2) and Tedrake (a, Chapter 3) for excellent explanations of the mechanics and theoretical foundations of producing motion in rigid bodies.

Considering the (toy) example presented in Figure 6a, then we can analytically write the end-effector’s position $p \in \mathbb{R}^2$ as a function of the robot’s configuration, $p = p(q), p : \mathcal{Q} \mapsto \mathbb{R}^2$. In particular, we have:

$$p(q) = \begin{pmatrix} p_x(\theta_1, \theta_2) \\ p_y(\theta_1, \theta_2) \end{pmatrix} = \begin{pmatrix} l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) \\ l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) \end{pmatrix} \in S_{l_1+l_2}^{n=2} = \{p(q) \in \mathbb{R}^2 : \|p(q)\|_2^2 \leq (l_1 + l_2)^2\} \forall q \in \mathcal{Q}$$

Deriving the end-effector’s *pose* $p \in \mathcal{P} \subset \mathbb{R}^{\frac{n(n+1)}{2}}$ (position and orientation) starting from the configuration $q \in \mathcal{Q} \subset \mathbb{R}^n$ of a n -joints robot is referred to as *forward kinematics* (FK), whereas identifying the configuration corresponding to any given target pose is termed *inverse kinematics* (IK). In that, FK is used to map a robot configuration into the corresponding end-effector pose, whereas IK is used to reconstruct the configuration(s) given an end-effector pose.

In the simplified case here considered (for which $p \equiv p$, as the orientation of the end-effector is disregarded for simplicity), one can solve the problem of controlling the end-effector’s location to reach a goal position p^* by solving analytically for $q : p(q) = f_{\text{FK}}(q) = p^*$. However, in the general case, one might not be able to solve this problem analytically, and can typically resort to iterative optimization methods comparing candidate solutions using a loss function (in the simplest case, $\|p(q) - p^*\|_2^2$ is a natural candidate), yielding:

$$\min_{q \in \mathcal{Q}} \|p(q) - p^*\|_2^2 \quad (1)$$

Analytical solutions are even less appealing when one considers the presence of obstacles in the robot’s workspace, resulting in constraints on the possible values of $q \in \mathcal{Q} \subset [-\pi, +\pi]^n \subset \mathbb{R}^n$ in the general case of n -links robots.

For instance, the robot in Figure 6b is (very naturally) obstructed by the presence of the surface upon which it rests: θ_1 can now exclusively vary within $[0, \pi]$, while possible variations in θ_2 depend on θ_1 (when $\theta_1 \rightarrow 0$ or $\theta_1 \rightarrow \pi$, further downwards movements are restricted). Even for a simplified kinematic model, developing techniques to solve 1 is in general non-trivial in the presence of constraints, particularly considering that the feasible set of solutions \mathcal{Q} may change across problems. Figure 6c provides an example of how the environment influences the feasible set considered, with a new set of constraints deriving from the position of a new obstacle.

Further, IK—solving 1 for a feasible q —only proves useful in determining information regarding the robot’s configuration in the goal pose, and crucially does not provide information on the *trajectory* to follow over time to reach a target pose. *Differential* inverse kinematics (diff-IK) complements IK by enabling trajectory tracking. Let $J(q)$ denote the Jacobian matrix of (partial) derivatives of the FK-function $f_{\text{FK}} : \mathcal{Q} \mapsto \mathcal{P}$, such that $J(q) = \frac{\partial f_{\text{FK}}(q)}{\partial q}$. Then, one can apply the chain rule to any $p(q) = f_{\text{FK}}(q)$ deriving $\dot{p} = J(q)\dot{q}$, finally relating variations in the robot configurations to variations in pose, thereby providing a platform for control.

Given a desired end-effector trajectory $\dot{p}^*(t)$ (1) indicating anchor regions in space and (2) how much time to spend in each region, diff-IK finds $\dot{q}(t)$ solving for joints' *velocities* instead of *configurations*,

$$\dot{q}(t) = \arg \min_{\nu} \|J(q(t))\nu - \dot{p}^*(t)\|_2^2 \quad (2)$$

Unlike 1, solving for \dot{q} is much less dependent on the environment (typically, variations in velocity are constrained by physical limits on the actuators represented with box-like constraints in 2). Conveniently, 2 also admits the closed-form solution, $\dot{q} = J(q)^+ \dot{p}^*$, where $J^+(q)$ denotes the Moore-Penrose pseudo-inverse of $J(q)$. Finally, discrete-time trajectories can then be obtained using forward integration, $q_{t+1} = q_t + \Delta t \dot{q}_t$ for a given Δt .

Following trajectories with diff-IK is a valid option in well-controlled and static environments (e.g., industrial manipulators in controlled manufacturing settings), and relies on the ability to define a set of target velocities to track $[\dot{p}_0^*, \dot{p}_1^*, \dots, \dot{p}_k^*]$ —an error-prone task largely carried out by human experts. Furthermore, diff-IK relies on the ability to (1) access $J(q) \forall q \in \mathcal{Q}$ and (2) compute its pseudo-inverse at every iteration of a given control cycle—a challenging assumption in highly dynamical settings, or for complex FK (e.g., nonlinear dynamics).

2.3.1 Adding Feedback Loops

While very effective when a goal trajectory has been well specified, the performance of diff-IK can degrade significantly in the presence of modeling/tracking errors, or in the presence of non-modeled dynamics in the environment.

One such case is presented in Figure 7, where another rigid body other than the manipulator is moving in the environment along the horizontal axis, with velocity \dot{x}_B . Accounting analytically for the presence of this disturbance—for instance, to prevent the midpoint of l_1 from ever colliding with the object—requires access to \dot{x}_B at least, to derive the equation characterizing the motion of the environment.

Less predictable disturbances however (e.g., $\dot{x}_B \leftarrow \dot{x}_B + \varepsilon, \varepsilon \sim N(0, 1)$) may prove challenging to model analytically, and one could attain the same result of preventing link-object collision by adding a condition on the distance between the midpoint of l_1 and x_B , enforced through a feedback loop on the position of the robot and object at each control cycle.

To mitigate the effect of modeling errors, sensing noise and other disturbances, classical pipelines indeed do augment diff-IK with feedback control looping back quantities of interest. In practice, following a trajectory with a closed feedback loop might consist in backwarding the error between the target and measured pose, $\Delta p = p^* - p(q)$, hereby modifying the control applied to $\dot{q} = J(q)^+ (\dot{p}^* + k_p \Delta p)$, with k_p defined as the (proportional) gain.

More advanced techniques for control consisting in feedback linearization, PID control, Linear Quadratic Regulator (LQR) or Model-Predictive Control (MPC) can be employed to stabilize tracking and reject moderate perturbations ((Siciliano and Khatib, 2016, Chapter 8) for in-detail tractation of this subject, or (Tedrake, a, Chapter 8) for a simple example in the case of a point-mass system). Nonetheless, feedback control presents its challenges as well: tuning gains remains laborious and system-specific. In particular, manipulation tasks present intermittent contacts inducing hybrid dynamics (mode switches) and discontinuities in the effective Jacobian, challenging the stability guarantees of the controller and thus often necessitating rather conservative gains and substantial hand-tuning.

We point the interested reader to Siciliano and Khatib (2016, Chapter 2,7,8), Lynch and Park (2017, Chapter 6,11), and Tedrake (a, Chapter 3,8) for extended coverage of FK, IK, diff-IK and control for (diff)-IK.

2.4 Limitations of Dynamics-based Robotics

Despite the last 60+ years of robotics research, autonomous robots are still largely incapable of performing tasks at human-level performance in the physical world across (1) robot embodiments (different manipulators, different locomotion platforms, etc.) and (2) tasks (tying shoe-laces, manipulating a diverse set of objects). While essential in the early development of robotics, the aforementioned methods require significant human expertise to be used in practice, and are typically specific to a particular applicative problem.

Other limitations of dynamics-based robotics follow.

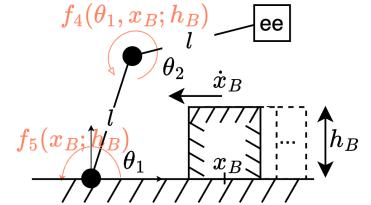
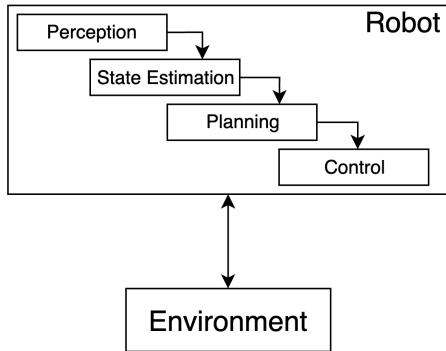
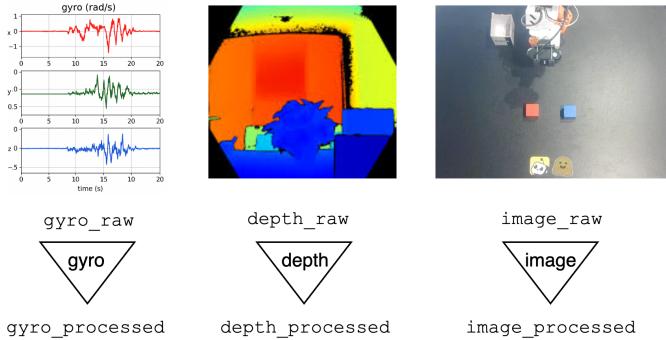


Figure 7 | Planar manipulator robot in the presence of a moving obstacle.

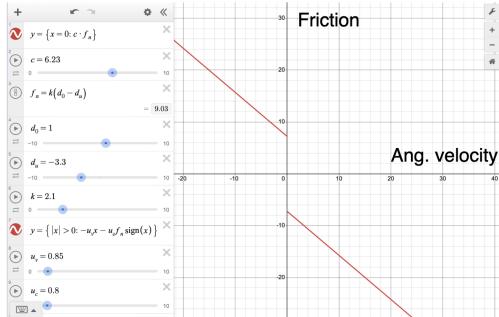
① Integration challenges



② Multimodality challenges



③ Undermodeling issues



Friction model for a manipulation task

④ Ignoring open-data growth

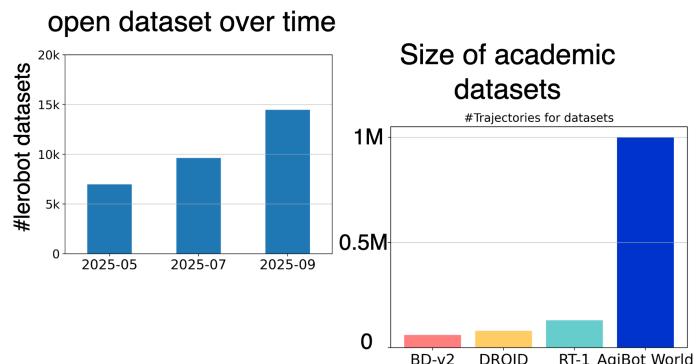


Figure 8 | Dynamics-based approaches to robotics suffer from several limitations: (1) orchestrating multiple components poses integration challenges; (2) the need to develop custom processing pipelines for the sensing modalities and tasks considered hinders scalability; (3) simplified analytical models of physical phenomena (here friction at the gripper; credits to [Antonova et al. \(2017\)](#)) limit real-world performance. Lastly, (4) dynamics-based methods overlook trends in the availability and growth of robotics data.

Dynamics-based robotics pipelines have historically been developed sequentially, engineering the different blocks now within most architectures for specific purposes. That is, sensing, state estimation, mapping, planning, (diff-)IK, and low-level control have been traditionally developed as distinct modules with fixed interfaces. Pipelining these specific modules proved error-prone, and brittleness emerges—alongside error compounding—whenever changes incur (e.g., changes in lighting for sensing, occlusion/failure of sensors, control failures). Adapting such a stack to new tasks or robotic platforms often entails re-specifying objectives, constraints, and heuristics at multiple stages, incurring significant engineering overhead.

Moreover, classical planners operate on compact, assumed-sufficient state representations; extending them to reason directly over raw, heterogeneous and noisy data streams is non-trivial. This results in a limited scalability to multimodal data and multitask settings, as incorporating high-dimensional perceptual inputs (RGB, depth, tactile, audio) traditionally required extensive engineering efforts to extract meaningful features for control. Also, the large number of tasks, coupled with the adoption of *per-task* planners, goal parameterizations, and safety constraints, results in an explosion in design and validation options, with little opportunity to reuse solutions across tasks.

Setting aside integration and scalability challenges: developing accurate modeling of contact, friction, and compliance for complicated remains difficult. Rigid-body approximations are often insufficient in the presence of deformable objects, and relying on approximated models hinders real-world applicability of the methods developed. In the case of complex and/or non-linear dynamics, even moderate mismatches in contact parameters, unmodeled flexibilities, or grasp-induced couplings can qualitatively affect the observed dynamics. Critically, even if a perfect model of the world was available, explicitly modeling the time-dependant effects of payload changes, tool substitutions, temperature-dependent friction, and component-aging on the resulting dynamics is, in most cases, simply unfeasible.

Lastly, dynamics-based methods (naturally) overlook the rather recent increase in availability of openly-available robotics datasets. The curation of academic datasets by large centralized groups of human experts in robotics (Ope; DRO; AgiBot-World-Contributors et al., 2025) is now increasingly complemented by a growing number of robotics datasets contributed in a decentralized fashion by individuals with varied expertises. If not tangentially, dynamics-based approaches are not posed to maximally benefit from this trend, which holds the premise of allowing generalization in the space of tasks and embodiments, like data was the cornerstone for advancements in vision (Alayrac et al., 2022) and natural-language understanding (Brown et al., 2020).

Taken together, these limitations (Figure 8) motivate the exploration of learning-based approaches that can (1) integrate perception and control more tightly, (2) adapt across tasks and embodiments with reduced expert intervention and (3) scale gracefully in performance as more robotics data becomes available.

3 Robot Learning

Approximate the solution, not the problem [...]

Richard Sutton

TL;DR

The need for expensive high-fidelity simulators can be obviated by learning from real-world data, using sample-efficient algorithms that can safely train directly on hardware.

Learning-based techniques for robotics naturally address the limitations presented in 2.1 (Figure 9). Learning-based techniques typically rely on prediction-to-action (*visuomotor policies*), thereby directly mapping sensorimotor inputs to predicted actions, streamlining control policies by removing the need to interface multiple components. Mapping sensorimotor inputs to actions directly also allows to add diverse input modalities, leveraging the automatic feature extraction characteristic of most modern learning systems. Further, learning-based approaches can in principle entirely bypass modeling efforts and instead rely exclusively on interactions data, proving transformative when dynamics are challenging to model or even entirely unknown. Lastly, learning for robotics (*robot learning*) is naturally well posed to leverage the growing amount of robotics data openly available, just as computer vision first and natural language processing later did historically benefit from large scale corpora of (possibly non curated) data, in great part overlooked by dynamics-based approaches.

Being a field at its relative nascent stages, no prevalent technique(s) proved distinctly better better in robot learning.

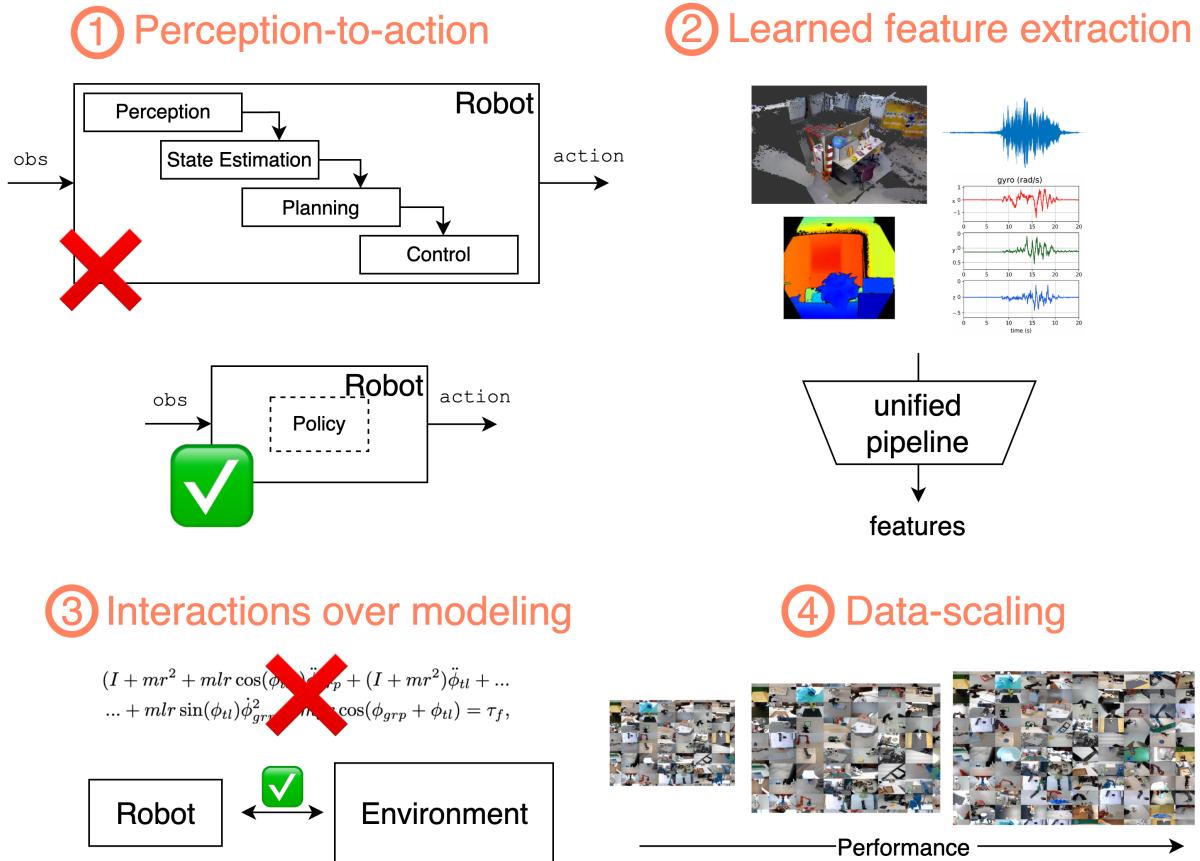


Figure 9 | Learning-based robotics streamlines perception-to-action by learning a (1) unified high-level controller capable to take (2) high-dimensional, unstructured sensorimotor information. Learning (3) does not require a dynamics model and instead focuses on interaction data, and (4) empirically correlates with the scale of the data used.

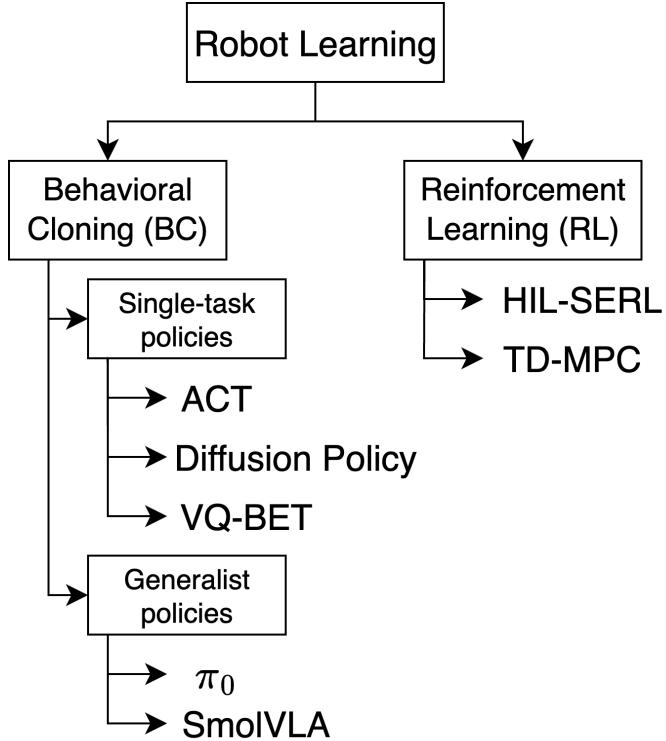


Figure 10 | Overview of the robot learning methods implemented in `1erobot`.

Still, two major classes of methods gained prominence: **reinforcement learning (RL)** and **Behavioral Cloning (BC)** (Figure 10). In this section, we provide a conceptual overview of applications of the former to robotics, as well as introduce practical examples of how to use RL within `1erobot`. We then introduce the major limitations RL suffers from, and introduce BC in the next section.

In Figure 10 we decided to include generalist robot models (Black et al., 2024; Shukor et al., 2025) alongside task-specific BC methods. While significant different in spirit—*generalist* models are language-conditioned and use instructions to generate motion valid across many tasks, while *task-specific* models are typically not language-conditioned and used to perform a single task—foundation models are largely trained to reproduce trajectories contained in a large training set of input demonstrations. Thus, we argue generalist policies can indeed be grouped alongside other task-specific BC methods, as they both leverage similar training data and schemas.

Figure 10 illustrates this categorization graphically, explicitly listing all the robot learning policies currently available in `1erobot`: Action Chunking with Transformers (ACT) (Zhao et al., 2023), Diffusion Policy (Chi et al., 2024), Vector-Quantized Behavior Transformer (VQ-BeT) (Lee et al., 2024), π_0 (Black et al., 2024), SmoVLA (Shukor et al., 2025), Human-in-the-loop Sample-efficient RL (HIL-SERL) (Luo et al., 2024) and TD-MPC (Hansen et al., 2022).

3.1 RL for Robotics

Applications of RL to robotics have been long studied, to the point the relationship between these two disciplines has been compared to that between physics and mathematics (Kober et al.). Indeed, due to their interactive and sequential nature, many robotics problems can be directly mapped to RL problems. Figure 11 depicts two of such cases. Reaching for an object to move somewhere else in the scene is an indeed sequential problem where at each cycle the controller needs to adjust the position of the robotic arm based on their current configuration and the (possibly varying) position of the object. Figure 11 also shows an example of a locomotion problem, where sequentiality is inherent in the problem formulation. While sliding to the side, the controller has to constantly keep adjusting to the robot’s proprioception to avoid failure (falling).

3.1.1 A (Concise) Introduction to RL

The RL framework (Sutton and Barto, 2018), which we briefly introduce here, has often been used to model robotics problems (Kober et al.). RL is a subfield within ML fundamentally concerned with the development of autonomous

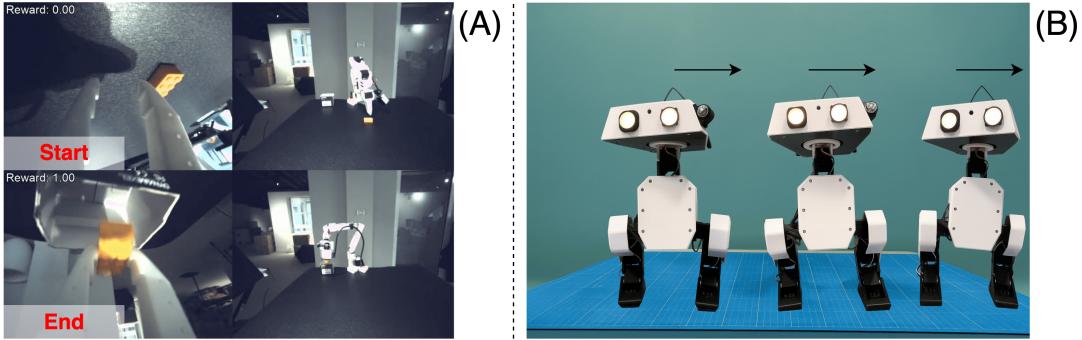


Figure 11 | Examples of two different robotics tasks performed using RL. In the manipulation task (A) an agent learns to reach for a yellow plastic block in its environment, and to put it inside of a box. In the locomotion task (B) an agent learns to move its center of mass sideways without falling.

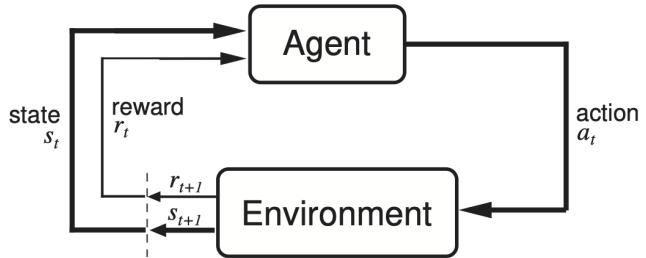


Figure 12 | Agent-Environment interaction diagram (image credits to Sutton and Barto (2018)).

systems (*agents*) learning how to *continuously behave* in an evolving environment, developing (ideally, well-performing) control strategies (*policies*). Crucially for robotics, RL agents can improve via trial-and-error only, thus entirely bypassing the need to develop explicit models of the problem dynamics, and rather exploiting interaction data only. In RL, this feedback loop (Figure 12) between actions and outcomes is established through the agent sensing a scalar quantity (*reward*).

Formally, interactions between an agent and its environment are typically modeled via a Markov Decision Process (MDP) (Bellman, 1957). Representing robotics problems via MDPs offers several advantages, including (1) incorporating uncertainty through MDP's inherently stochastic formulation and (2) providing a theoretically sound framework for learning *without* an explicit dynamic model. While accommodating also a continuous time formulation, MDPs are typically considered in discrete time in RL, thus assuming interactions to atomically take place over the course of discrete *timestep* $t = 0, 1, 2, 3, \dots, T$. MDPs allowing for an unbounded number of interactions ($T \rightarrow +\infty$) are typically termed *infinite-horizon*, and opposed to *finite-horizon* MDPs in which T cannot grow unbounded. Unless diversely specified, we will only be referring to discrete-time finite-horizon (*episodic*) MDPs here.

Formally, a lenght- T Markov Decision Process (MDP) is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{D}, r, \gamma, \rho, T \rangle$, where:

- \mathcal{S} is the *state space*; $s_t \in \mathcal{S}$ denotes the (possibly non-directly observable) environment state at time t . In robotics, states often comprise robot configuration and velocities (q_t, \dot{q}_t), and can accomodate sensor readings such as camera or audio streams.
- \mathcal{A} is the *action space*; $a_t \in \mathcal{A}$ may represent joint torques, joint velocities, or even end-effector commands. In general, actions correspond to commands intervening on the configuration of the robot.
- \mathcal{D} represents the (possibly non-deterministic) environment dynamics, with $\mathcal{D} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ corresponding to $\mathcal{D}(s_t, a_t, s_{t+1}) = \mathbb{P}(s_{t+1}|s_t, a_t)$. For instance, for a planar manipulator dynamics could be considered deterministic when the environment is fully described (Figure 6a), and stochastic when unmodeled disturbances depending on non-observable parameters intervene (Figure 7).
- $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is the *reward function*, weighing the transition (s_t, a_t, s_{t+1}) in the context of the achievement of an arbitrary goal. For instance, a simple reward function for quickly moving the along the x axis in 3D-space (Figure 11) could be based on the absolute position of the robot along the x axis (p_x), present negative penalties for falling over (measured from p_z) and a introduce bonuses \dot{p}_x for speed, $r(s_t, a_t, s_{t+1}) \equiv r(s_t) = p_{x_t} \cdot \dot{p}_{x_t} - \frac{1}{p_{z_t}}$.

Lastly, $\gamma \in [0, 1]$ represent the discount factor regulating preference for immediate versus long-term reward (with an effective horizon equal to $\frac{1}{1-\gamma}$), and ρ is the distribution, defined over \mathcal{S} , the MDP's *initial* state is sampled from, $s_0 \sim \rho$.

A length- T *trajectory* is the (random) sequence

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_{T-1}, a_{T-1}, r_{T-1}, s_T), \quad (3)$$

with per-step rewards defined as $r_t = r(s_t, a_t, s_{t+1})$ for ease of notation. Interestingly, assuming both the environment dynamics and conditional distribution over actions given states—the *policy*—to be *Markovian*:

$$\mathbb{P}(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_0, a_0) = \mathbb{P}(s_{t+1}|s_t, a_t) \quad (4)$$

$$\mathbb{P}(a_t|s_t, a_{t-1}, s_{t-1}, s_0, a_0) = \mathbb{P}(a_t|s_t) \quad (5)$$

The probability of observing a given trajectory τ factorizes into

$$\mathbb{P}(\tau) = \mathbb{P}(s_0) \prod_{t=0}^{T-1} \mathbb{P}(s_{t+1}|s_t, a_t) \mathbb{P}(a_t|s_t). \quad (6)$$

Policies $\mathbb{P}(a_t|s_t)$ are typically indicated as $\pi(a_t|s_t)$, and often parametrized via θ , yielding $\pi_\theta(a_t|s_t)$. Policies are trained optimizing the (discounted) *return* associated to a given τ , i.e. the (random) sum of measured rewards over trajectory:

$$G(\tau) = \sum_{t=0}^{T-1} \gamma^t r_t.$$

In that, agents seek to learn control strategies (*policies*, π_θ) maximizing the expected return $\mathbb{E}_{\tau \sim \pi_\theta} G(\tau)$. For a given dynamics \mathcal{D} —i.e., for a given problem—taking the expectation over the (possibly random) trajectories resulting from acting according to a certain policy provides a direct, goal-conditioned ordering in the space of all the possible policies Π , yielding the (maximization) target $J : \Pi \mapsto \mathbb{R}$

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim \mathbb{P}_{\theta; \mathcal{D}}} [G(\tau)], \quad (7)$$

$$\mathbb{P}_{\theta; \mathcal{D}}(\tau) = \rho \prod_{t=0}^{T-1} \mathcal{D}(s_t, a_t, s_{t+1}) \pi_\theta(a_t|s_t). \quad (8)$$

Because in the RL framework the agent is assumed to only be able to observe the environment dynamics and not to intervene on them, 7 varies exclusively with the policy followed. In turn, MDPs naturally provide a framework to optimize over the space of the possible behaviors an agent might enact ($\pi \in \Pi$), searching for the *optimal policy* $\pi^* = \arg \max_\theta J(\pi_\theta)$, where θ is the parametrization adopted by the policy set $\Pi : \pi_\theta \in \Pi, \forall \theta$. Other than providing a target for policy search, $G(\tau)$ can also be used as a target to discriminate between states and state-action pairs. Given any state $s \in \mathcal{S}$ —e.g., a given configuration of the robot—the *state-value* function

$$V_\pi(s) = \mathbb{E}_{\tau \sim \pi} [G(\tau) | s_0 = s]$$

can be used to discriminate between desirable and undesirable state in terms of long-term (discounted) reward maximization, under a given policy π . Similarly, the *state-action* value function also conditions the cumulative discounted reward on selecting action a when in s , and thereafter act according to π :

$$Q_\pi(s, a) = \mathbb{E}_{\tau \sim \pi} [G(\tau) | s_0 = s, a_0 = a]$$

Crucially, value functions are interrelated:

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1} \sim \mathbb{P}(\bullet|s_t, a_t)} [r_t + \gamma V_\pi(s_{t+1})] \quad (9)$$

$$V_\pi(s_t) = \mathbb{E}_{a_t \sim \pi(\bullet|s_t)} [Q_\pi(s_t, a_t)] \quad (10)$$

Inducing an ordering over states and state-action pairs under π , value functions are central to most RL algorithms. A variety of methods have been developed in RL as standalone attempts to find (approximate) solutions to the problem of maximizing cumulative reward (Figure 13).

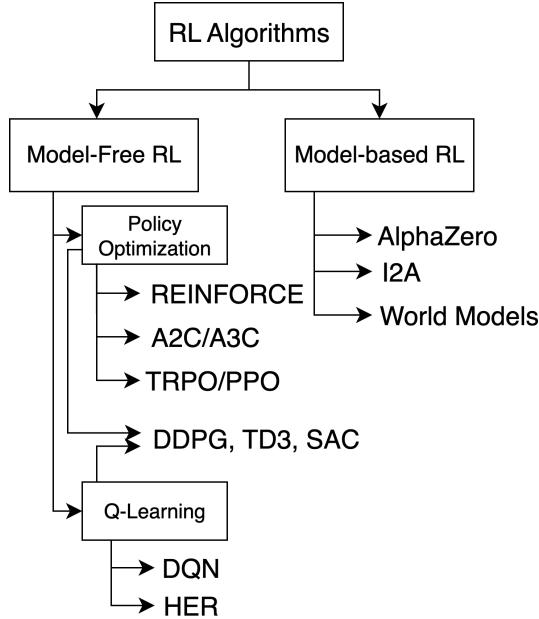


Figure 13 | Popular RL algorithms. See Achiam (2018) for a complete list of citations.

Popular approaches to continuous state and action space—such as those studied within robotics—include Schulman et al. (2017a,b); Haarnoja et al. (2018). Across manipulation (Akkaya et al., 2019) and locomotion (Lee et al., 2020) problems, RL proved extremely effective in providing a platform to (1) adopt a unified, streamlined perception-to-action pipeline, (2) natively integrate proprioception with multi-modal high-dimensional sensor streams (3) disregard a description of the environment dynamics, by focusing on observed interaction data rather than modeling, and (4) anchor policies in the experience collected and stored in datasets. For a more complete survey of applications of RL to robotics, we refer the reader to Kober et al.; Tang et al. (2024).

3.1.2 Real-world RL for Robotics

Streamlined end-to-end control pipelines, data-driven feature extraction and a disregard for explicit modeling in favor of interaction data are all features of RL for robotics. However, particularly in the context of real-world robotics, RL still suffers from limitations concerning machine safety and learning efficiency.

First, especially early in training, **actions are typically explorative, and thus erratic**. On physical systems, untrained policies may command high velocities, self-colliding configurations, or torques exceeding joint limits, leading to wear and potential hardware damage. Mitigating these risks requires external safeguards (e.g., watchdogs, safety monitors, emergency stops), often incurring in a high degree of human supervision. Further, in the typical episodic setting considered in most robotics problems, experimentation is substantially slowed down by the need to manually reset the environment over the course of training, a time-consuming and brittle process.

Second, learning with a limited number of samples remains problematic in RL, **limiting the applicability of RL in real-world robotics due to consequently prohibitive timescales of training**. Even strong algorithms such as SAC (Haarnoja et al., 2018) typically require a large numbers of transitions $\{(s_t, a_t, r_t, s_{t+1})\}_{t=1}^N$. On hardware, generating these data is time-consuming and can even be prohibitive.

Training RL policies in simulation (Tobin et al., 2017) addresses both issues: it eliminates physical risk and dramatically increases throughput. Yet, simulators require significant modeling effort, and rely on assumptions (simplified physical modeling, instantaneous actuation, static environmental conditions, etc.) limiting transferring policies learned in simulation due the discrepancy between real and simulated environments (*reality gap*, Figure 14). **Domain randomization** (DR) is a popular technique to overcome the reality gap, consisting in randomizing parameters of the simulated environment during training, to induce robustness to specific disturbances. In turn, DR is employed to increase the diversity of scenarios over the course of training, improving on the chances sim-to-real transfer (Akkaya et al., 2019; Antonova et al., 2017; Ji et al., 2023). In practice, DR is performed further parametrizing the *simulator*'s dynamics $\mathcal{D} \equiv \mathcal{D}_\xi$ with a *dynamics* (random) vector ξ drawn an arbitrary distribution, $\xi \sim \Xi$. Over the course of training—typically at each episode's reset—a new ξ is drawn, and used to specify the environment's dynamics for

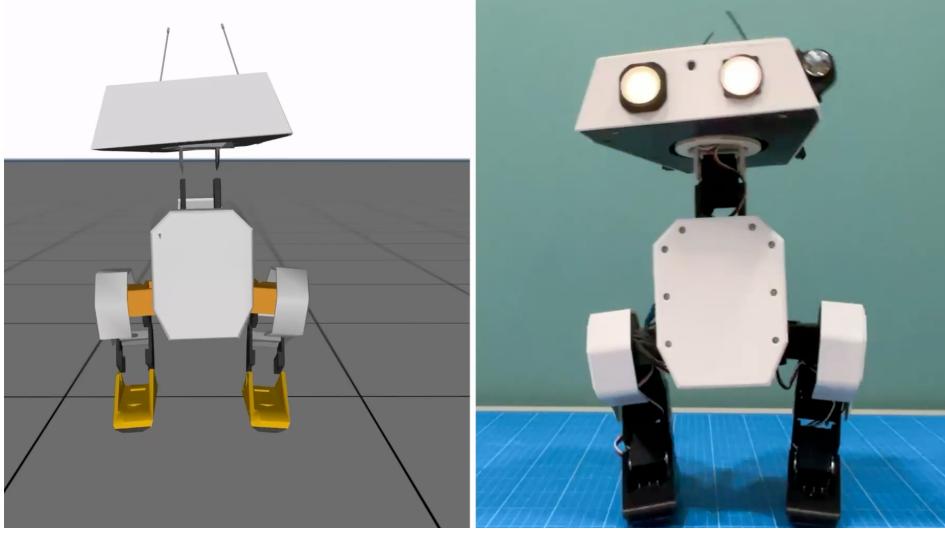


Figure 14 | Simulated (left) vs. real-world (right) OpenDuck. Discrepancies in the simulation dynamics (*reality gap*) pose risks to policy transfer.

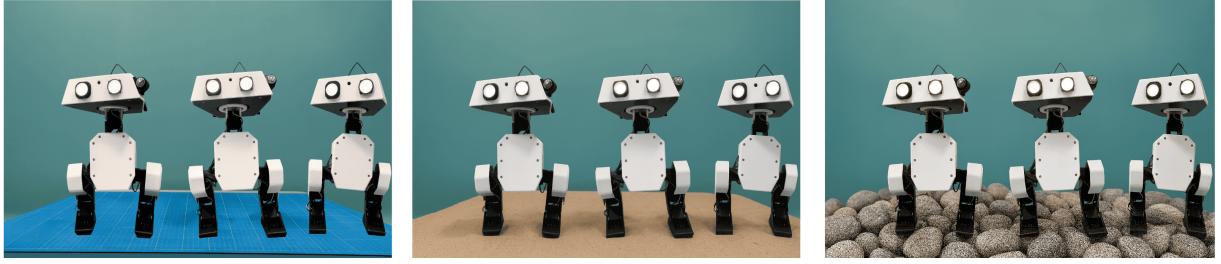


Figure 15 | The same locomotion task can be carried out in different (simulated) domains (exemplified by the difference in terrains) at training time, resulting to increased robustness over diverse environment dynamics.

that episode. For instance, one could decide to randomize the friction coefficient of the surface in a locomotion task (Figure 15), or the center of mass of an object for a manipulation task.

While effective in transferring policies across the reality gap in real-world robotics (Tobin et al., 2017; Akkaya et al., 2019; Ji et al., 2023; Tiboni et al., 2024), DR often requires extensive manual engineering. First, identifying which parameters to randomize—i.e., the *support* $\text{supp}(\Xi)$ of Ξ —is an inherently task specific process. When locomoting over different terrains, choosing to randomize the friction coefficient is a reasonable choice, yet not completely resolutive as other factors (lightning conditions, external temperature, joints’ fatigue, etc.) may prove just as important, making selecting these parameters yet another source of brittleness.

Selecting the dynamics distribution Ξ is also non-trivial. On the one hand, distributions with low entropy might risk to cause failure at transfer time, due to the limited robustness induced over the course of training. On the other hand, excessive randomization may cause over-regularization and hinder performance. Consequently, the research community investigated approaches to automatically select the randomization distribution Ξ , using signals from the training process or tuning it to reproduce observed real-world trajectories. Akkaya et al. (2019) use a parametric uniform distribution $\mathcal{U}(a, b)$ as Ξ , widening the bounds as training progresses and the agent’s performance improves (AutoDR). While effective, AutoDR requires significant tuning—the bounds are widened by a fixed, pre-specified amount Δ —and may disregard data when performance *does not* improve after a distribution update (Tiboni et al., 2024). Tiboni et al. (2024) propose a similar method to AutoDR (DORAEMON) to evolve Ξ based on training signal, but with the key difference of explicitly maximizing the entropy of parametric Beta distributions, inherently more flexible than uniform distributions. DORAEMON proves particularly effective at dynamically increasing the entropy levels of the training distribution by employing a max-entropy objective, under performance constraints formulation. Other approaches to automatic DR consist in specifically tuning Ξ to align as much as possible the simulation and real-world domains. For instance, Chebotar et al. (2019) interleave in-simulation policy training with repeated real-world policy rollouts used to adjust Ξ based on real-world data, while Tiboni et al. (2023) leverage a

single, pre-collected set of real-world trajectories and tune Ξ under a simple likelihood objective.

While DR has shown promise, it does not address the main limitation that, even under the assumption that an ideal distribution Ξ to sample from was indeed available, many robotics problems **cannot be simulated with high-enough fidelity under practical computational constraints** in the first place. Simulating contact-rich manipulation of possibly deformable or soft materials—i.e., *folding a piece of clothing*—can be costly and even time-intensive, limiting the benefits of in-simulation training.

A perhaps more fundamental limitation of RL for robotics is the general unavailability of complicated tasks' *dense* reward function, the design of which is essentially based on human expertise and trial-and-error. In practice, *sparse* reward functions can be used to conclude whether one specific goal has been attained—*has this t-shirt been correctly folded?*—but unfortunately incur in more challenging learning. As a result, despite notable successes, deploying RL directly on real-world robots at scale remains challenging.

3.1.3 RL in lerobot: sample-efficient, data-driven, and real-world

To make the most of (1) the growing number of openly available datasets and (2) relatively inexpensive robots like the SO-100, RL could (1) be anchored in already-collected trajectories—limiting erratic and dangerous exploration—and (2) train in the real-world directly—bypassing the aforementioned issues with low-fidelity simulations. In such a context, sample-efficient learning is also paramount, as training on the real-world is inherently time-bottlenecked.

Off-policy algorithms like Soft Actor-Critic (SAC) (Haarnoja et al., 2018) tend to be more sample efficient than their on-policy counterpart (Schulman et al., 2017b), due to the presence a *replay buffer* used over the course of the training. Other than allowing to re-use transitions (s_t, a_t, r_t, s_{t+1}) over the course of training, the replay buffer can also accomodate for the injection of previously-collected data in the training process (Ball et al., 2023). Using expert demonstrations to guide learning together with learned rewards, RL training can effectively be carried out in the real-world (Luo et al., 2025). Interestingly, when completed with in-training human interventions, real-world RL agents have been shown to learn policies with near-perfect success rates on challenging manipulation tasks in 1-2 hours (Luo et al., 2024).

Sample-efficient RL In an MDP, the optimal policy π^* can be derived from its associated Q -function, Q_{π^*} , and in particular the optimal action(s) $\mu(s_t)$ can be selected maximizing the optimal Q -function over the action space,

$$\mu(s_t) = \max_{a_t \in \mathcal{A}} Q_{\pi^*}(s_t, a_t).$$

Interestingly, the Q^* -function satisfies a recursive relationship (*Bellman equation*) based on a very natural intuition¹:

[...] If the optimal value $Q^*(s_{t+1}, a_{t+1})$ of the [state] s_{t+1} was known for all possible actions a_{t+1} , then the optimal strategy is to select the action a_{t+1} maximizing the expected value of $r_t + \gamma Q^*(s_{t+1}, a_{t+1})$

$$Q^*(s_t, a_t) = \mathbb{E}_{s_{t+1} \sim \mathbb{P}(\bullet|s_t, a_t)} \left[r_t + \gamma \max_{a_{t+1} \in \mathcal{A}} Q^*(s_{t+1}, a_{t+1}) \middle| s_t, a_t \right]$$

In turn, the optimal Q -function is guaranteed to be self-consistent by definition. *Value-iteration* methods exploit this relationship (and/or its state-value counterpart, $V^*(s_t)$) by iteratively updating an initial estimate of Q^* , Q_k using the Bellman equation as update rule (*Q-learning*):

$$Q_{i+1}(s_t, a_t) \leftarrow \mathbb{E}_{s_{t+1} \sim \mathbb{P}(\bullet|s_t, a_t)} \left[r_t + \gamma \max_{a_{t+1} \in \mathcal{A}} Q_i(s_{t+1}, a_{t+1}) \middle| s_t, a_t \right], \quad i = 0, 1, 2, \dots, K$$

Then, one can derive the (ideally, near-optimal) policy by explicitly maximizing over the action space the final (ideally, near-optimal) estimate $Q_K \approx Q^*$ at each timestep. In fact, under certain assumptions on the MDP considered, $Q_K \rightarrow Q^*$ as $K \rightarrow \infty$.

Effective in its early applications to small-scale discrete problems and theoretically sound, vanilla Q-learning was found complicated to scale to large $\mathcal{S} \times \mathcal{A}$ problems, in which the storing of $Q : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ alone might result prohibitive. Also, vanilla Q-learning is not directly usable for *continuous*, unstructured state-action space MPDs, such as those considered in robotics. In their seminal work on *Deep Q-Learning* (DQN), Mnih et al. (2013) propose learning Q-values using deep convolutional neural networks, thereby accomodating for large and even unstructured

¹Quote from Mnih et al. (2013). The notation used has slightly been adapted for consistency with the rest of this tutorial.

state spaces. DQN parametrizes the Q-function using a neural network with parameters θ , updating the parameters by sequentially minimizing the expected squared temporal-difference error (TD-error, δ_i):

$$\mathcal{L}(\theta_i) = \mathbb{E}_{(s_t, a_t) \sim \chi(\bullet)} \left[\underbrace{(y_i - Q_{\theta_i}(s_t, a_t))^2}_{\delta_i} \right], \quad (11)$$

$$y_i = \mathbb{E}_{s_{t+1} \sim \mathbb{P}(\bullet|s_t, a_t)} \left[r_t + \gamma \max_{a_t \in \mathcal{A}} Q_{\theta_{i-1}}(s_{t+1}, a_{t+1}) \right], \quad (12)$$

Where χ represents a behavior distribution over state-action pairs. Crucially, χ can in principle be different from the policy being followed, effectively allowing to reuse prior data stored in a *replay buffer* in the form of (s_t, a_t, r_t, s_{t+1}) transitions, used to form the TD-target y_i , TD-error δ_i and loss function 11 via Monte-Carlo (MC) estimates.

While effective in handling large, unstructured state spaces for discrete action-space problems, DQN application's to continuous control problems proved challenging. Indeed, in the case of high-capacity function approximators such as neural networks, solving $\max_{a_t \in \mathcal{A}} Q_{\theta}(s_t, a_t)$ at each timestep is simply unfeasible due to the (1) continuous nature of the action space ($\mathcal{A} \subset \mathbb{R}^n$ for some n) and (2) impossibility to express the find a cheap (ideally, closed-form) solution to Q_{θ} . Silver et al. (2014) tackle this fundamental challenge by using a *deterministic* function of the state s_t as policy, $\mu_{\phi}(s_t) = a_t$, parametrized by ϕ . Thus, policies can be iteratively refined updating ϕ along the direction:

$$d_{\phi} = \mathbb{E}_{s_t \sim \mathbb{P}(\bullet)} [\nabla_{\phi} Q(s_t, a_t)|_{a_t=\mu_{\phi}(s_t)}] = \mathbb{E}_{s_t \sim \mathbb{P}(\bullet)} [\nabla_{a_t} Q(s_t, a_t)|_{a_t=\mu_{\phi}(s_t)} \cdot \nabla_{\phi} \mu(s_t)] \quad (13)$$

Provably, 13 is the *deterministic policy gradient* (DPG) of the policy μ_{ϕ} (Silver et al., 2014), so that updates $\phi_{k+1} \leftarrow \phi_k + \alpha d_{\phi}$ are guaranteed to increase the (deterministic) cumulative discounted reward, $J(\mu_{\phi})$. Lillicrap et al. (2019) extended DPG to the case of (1) high-dimensional unstructured observations and (2) continuous action spaces, introducing Deep Deterministic Policy Gradient (DDPG), an important algorithm RL and its applications to robotics. DDPG adopts a modified TD-target compared to the one defined in 12, by maintaining a policy network used to select actions, yielding

$$y_i = \mathbb{E}_{s_{t+1} \sim \mathbb{P}(\bullet|s_t, a_t)} [r_t + \gamma Q_{\theta_{i-1}}(s_{t+1}, \mu_{\phi}(s_{t+1}))]. \quad (14)$$

Similarly to DQN, DDPG also employs the same replay buffer mechanism, to reuse past transitions over training for increased sample efficiency and estimate the loss function via MC-estimates.

Soft Actor-Critic (SAC) (Haarnoja et al., 2018) is a derivation of DDPG in the max-entropy (MaxEnt) RL framework, in which RL agents are tasked with *maximizing the discounted cumulative reward, while acting as randomly as possible*. MaxEnt RL (Haarnoja et al., 2017) has proven particularly robust thanks to the development of diverse behaviors, incentivized by its entropy-regularization formulation. In that, MaxEnt revisits the RL objective $J(\pi)$ to specifically account for the policy entropy,

$$J(\pi) = \sum_{t=0}^T \mathbb{E}_{(s_t, a_t) \sim \chi} [r_t + \alpha \mathcal{H}(\pi(\bullet|s_t))] \quad (15)$$

This modified objective results in the *soft* TD-target:

$$y_i = \mathbb{E}_{s_{t+1} \sim \mathbb{P}(\bullet|s_t, a_t)} [r_t + \gamma (Q_{\theta_{i-1}}(s_{t+1}, a_{t+1}) - \alpha \log \pi_{\phi}(a_{t+1}|s_{t+1}))], \quad a_{t+1} \sim \pi_{\phi}(\bullet|s_t) \quad (16)$$

Similarly to DDPG, SAC also maintains an explicit policy, trained under the same MaxEnt framework for the maximization of 15, and updated using:

$$\pi_{k+1} \leftarrow \arg \min_{\pi' \in \Pi} D_{\text{KL}} \left(\pi'(\bullet|s_t) \middle\| \frac{\exp(Q_{\pi_k}(s_t, \bullet))}{Z_{\pi_k}(s_t)} \right) \quad (17)$$

The update rule provided in 17 optimizes the policy while projecting it on a set Π of tractable distributions (e.g., Gaussians, Haarnoja et al. (2017)).

Sample-efficient, data-driven RL Importantly, sampling (s_t, a_t, r_t, s_{t+1}) from the replay buffer D conveniently allows to approximate the previously introduced expectations for TD-target and TD-error through Monte-Carlo (MC) estimates. The replay buffer D also proves extremely useful in maintaining a history of previous transitions and using it for training, improving on sample efficiency. Furthermore, it also naturally provides an entry point to inject offline trajectories recorded, for instance, by a human demonstrator, into the training process.

Reinforcement Learning with Prior Data (RLPD) (Ball et al., 2023) is an Offline-to-Online RL algorithm leveraging prior data to effectively accelerate the training of a SAC agent. Unlike previous works on Offline-to-Online RL, RLPD avoids any pre-training and instead uses the available offline data D_{offline} to improve online-learning from scratch. During each training step, transitions from both the offline and online replay buffers are sampled in equal proportion, and used in the underlying SAC routine.

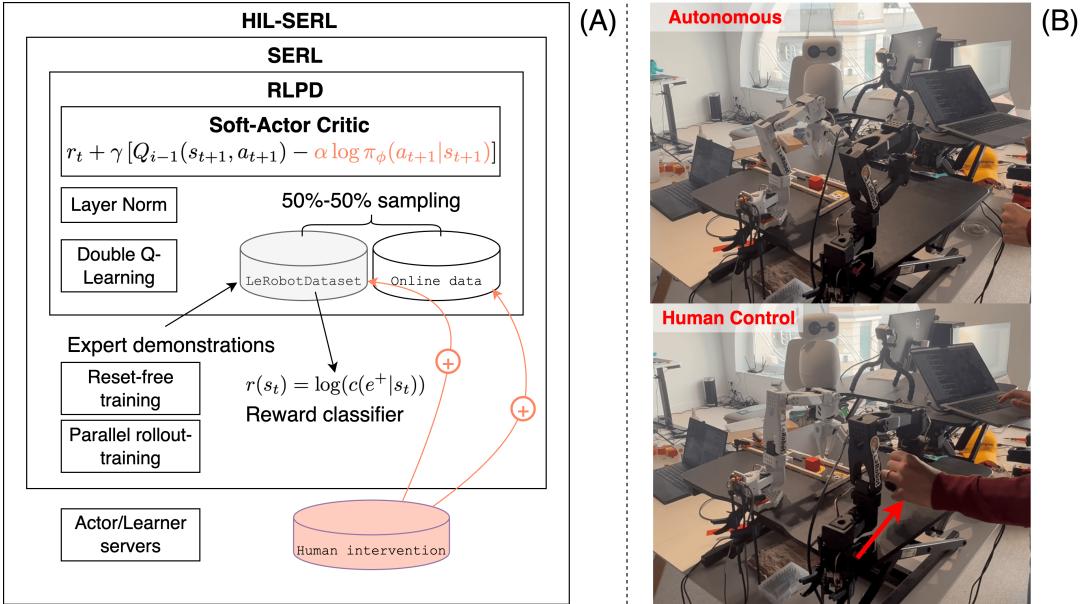


Figure 16 | (A) HIL-SERL allows for real-world training of high performance RL agents by building on top advancements presented by of SAC, RLPD and SERL. (B) Example of human intervention during a HIL-SERL training process on a SO-100.

Sample-efficient, data-driven, real-world RL Despite the possibility to leverage offline data for learning, the effectiveness of real-world RL training is still limited by the need to define a task-specific, hard-to-define reward function. Further, even assuming to have access to a well-defined reward function, typical robotics pipelines rely mostly on proprioceptive inputs augmented by camera streams of the environment. As such, even well-defined rewards would need to be derived from processed representations of unstructured observations, introducing brittleness. In their technical report, Luo et al. (2025) empirically address the needs (1) to define a reward function and (2) to use it on image observations, by introducing a series of tools to allow for streamlined training of *reward classifiers* c , as well as jointly learn forward-backward controllers to speed up real-world RL. Reward classifiers are particularly useful in treating complex tasks—e.g., folding a t-shirt—for which a precise reward formulation is arbitrarily complex to obtain, or that do require significant shaping and are more easily learned directly from demonstrations of success (e^+) or failure (e^-) states, $s \in \mathcal{S}$, with a natural choice for the state-conditioned reward function being $r: \mathcal{S} \mapsto \mathbb{R}$ being $r(s) = \log c(e^+ \text{ vert})$. Further, Luo et al. (2025) demonstrate the benefits of learning *forward* (executing the task from initial state to completion) and *backward* (resetting the environment to the initial state from completion) controllers, parametrized by separate policies.

Lastly, in order to improve on the robustness of their approach to different goals while maintaining practical scalability, Luo et al. (2025) introduced a modified state and action space, expressing proprioceptive configurations q and actions \dot{q} in the frame of end-effector pose at $t = 0$. Randomizing the initial pose of the end-effector (s_0), Luo et al. (2025) achieved a similar result to that of having to manually randomize the environment at every timestep, but with the benefit of maintaining the environment in the same condition across multiple training episodes, achieving higher scalability of their method thanks to the increased practicality of their approach.

Building on off-policy deep Q-learning with replay buffers, entropy regularization for better exploration and performance, expert demonstrations to guide learning, and a series of tools and recommendations for real-world training using reward classifiers (Figure 16), Luo et al. (2024) introduce human interactions during training, learning near-optimal policies in challenging real-world manipulation tasks in 1-2 hours.

Human in the Loop Sample Efficient Robot reinforcement Learning (HIL-SERL) (Luo et al., 2024) augments offline-to-online RL with targeted human corrections during training, and employs prior data to (1) train a reward classifier and (2) bootstrap RL training on expert trajectories. While demonstrations provide the initial dataset seeding learning and constraining early exploration, interactive corrections allow a human supervisor to intervene on failure modes and supply targeted interventions to aid the learning process. Crucially, human interventions are stored in both the offline and online replay buffers, differently from the autonomous transitions generated at training time and stored in the online buffer only. Consequently, given an intervention timestep $k \in (0, T)$, length- K human intervention data $\{s_k^{\text{human}}, a_k^{\text{human}}, r_k^{\text{human}}, s_{k+1}^{\text{human}}\}_{k=1}^K$ is more likely to be sampled for off-policy learning than the data generated online

during training, providing stronger supervision to the agent while still allowing for autonomous learning. Empirically, HIL-SERL attains near-perfect success rates on diverse manipulation tasks within 1-2 hours of training (Luo et al., 2024), underscoring how offline datasets with online RL can markedly improve stability and data efficiency, and ultimately even allow real-world RL-training.

3.1.4 Code Example: Real-world RL

TODO(fracapuano): work out rl training example

3.1.5 Limitations of RL in Real-World Robotics: Simulators and Reward Design

Despite the advancements in real-world RL training, solving robotics training RL agents in the real world still suffers from the following limitations:

- In those instances where real-world training experience is prohibitively expensive to gather (Degrave et al., 2022; Bellemare et al., 2020), in-simulation training is often the only option. However, high-fidelity simulators for real-world problems can be difficult to build and maintain, especially for contact-rich manipulation and tasks involving deformable or soft materials.
- Reward design poses an additional source of brittleness. Dense shaping terms are often required to guide exploration in long-horizon problems, but poorly tuned terms can lead to specification gaming or local optima. Sparse rewards avoid shaping but exacerbate credit assignment and slow down learning. In practice, complex behaviors require efforts shaping rewards: a brittle and error prone process.

Advances in Behavioral Cloning (BC) from corpora of human demonstrations address both of these concerns. By learning in a supervised fashion to reproduce expert demonstrations, BC methods prove competitive while bypassing the need for simulated environments and hard-to-define reward functions.

References

- DROID: A Large-Scale In-the-Wild Robot Manipulation Dataset. <https://droid-dataset.github.io/>.
- Open X-Embodiment: Robotic Learning Datasets and RT-X Models. <https://robotics-transformer-x.github.io/>.
- Joshua Achiam. Spinning up in deep reinforcement learning. 2018.
- AgiBot-World-Contributors, Qingwen Bu, Jisong Cai, Li Chen, Xiuqi Cui, Yan Ding, Siyuan Feng, Shenyuan Gao, Xindong He, Xuan Hu, Xu Huang, Shu Jiang, Yuxin Jiang, Cheng Jing, Hongyang Li, Jialu Li, Chiming Liu, Yi Liu, Yuxiang Lu, Jianlan Luo, Ping Luo, Yao Mu, Yuehan Niu, Yixuan Pan, Jiangmiao Pang, Yu Qiao, Guanghui Ren, Cheng Ruan, Jiaqi Shan, Yongjian Shen, Chengshi Shi, Mingkang Shi, Modi Shi, Chonghao Sima, Jianheng Song, Huijie Wang, Wenhao Wang, Dafeng Wei, Chengan Xie, Guo Xu, Junchi Yan, Cunbiao Yang, Lei Yang, Shukai Yang, Maoqing Yao, Jia Zeng, Chi Zhang, Qinglin Zhang, Bin Zhao, Chengyue Zhao, Jiaqi Zhao, and Jianchao Zhu. AgiBot World Colosseo: A Large-scale Manipulation Platform for Scalable and Intelligent Embodied Systems, August 2025.
- Pulkit Agrawal. Computational Sensorimotor Learning.
- Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving Rubik’s Cube with a Robot Hand, October 2019.
- Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Andrew Brock, Aida Nematizadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karen Simonyan. Flamingo: A Visual Language Model for Few-Shot Learning, November 2022.
- Jorge Aldaco, Travis Armstrong, Robert Baruch, Jeff Bingham, Sanky Chan, Debidatta Dwibedi, Chelsea Finn, Pete Florence, Spencer Goodrich, Wayne Gramlich, Alexander Herzog, Jonathan Hoech, Thinh Nguyen, Ian Storz, Baruch Tabanpour, Jonathan Tompson, Ayzaan Wahid, Ted Wahrburg, Sichun Xu, Sergey Yaroshenko, and Tony Z Zhao. ALOHA 2: An Enhanced Low-Cost Hardware for Bimanual Teleoperation.
- Mohammad Alizadeh and Zheng H. Zhu. A comprehensive survey of space robotic manipulators for on-orbit servicing. *Frontiers in Robotics and AI*, 11, October 2024. ISSN 2296-9144. doi: 10.3389/frobt.2024.1470950.
- Rika Antonova, Silvia Cruciani, Christian Smith, and Danica Kragic. Reinforcement Learning for Pivoting Task, March 2017.

Philip J. Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient Online Reinforcement Learning with Offline Data, May 2023.

Kostas E. Bekris, Joe Doerr, Patrick Meng, and Sumanth Tangirala. The State of Robot Motion Generation, October 2024.

Marc G. Bellemare, Salvatore Candido, Pablo Samuel Castro, Jun Gong, Marlos C. Machado, Subhodeep Moitra, Sameera S. Ponda, and Ziyu Wang. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836): 77–82, December 2020. ISSN 1476-4687. doi: 10.1038/s41586-020-2939-8.

Richard Bellman. A Markovian Decision Process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957. ISSN 0095-9057.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolò Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_{π_0} : A Vision-Language-Action Flow Model for General Robot Control, October 2024.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners, July 2020.

Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8973–8979. IEEE, 2019.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion, March 2024.

Jonathan H. Connell and Sridhar Mahadevan, editors. *Robot Learning*. Springer US, Boston, MA, 1993. ISBN 978-1-4613-6396-5 978-1-4615-3184-5. doi: 10.1007/978-1-4615-3184-5.

Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de las Casas, Craig Donner, Leslie Fritz, Cristian Galperti, Andrea Huber, James Keeling, Maria Tsimpoukelli, Jackie Kay, Antoine Merle, Jean-Marc Moret, Séb Noury, Federico Pesamosca, David Pfau, Olivier Sauter, Cristian Sommariva, Stefano Coda, Basil Duval, Ambrogio Fasoli, Pushmeet Kohli, Koray Kavukcuoglu, Demis Hassabis, and Martin Riedmiller. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, February 2022. ISSN 1476-4687. doi: 10.1038/s41586-021-04301-9.

Jun Fujita, Daisuke Soda, Chotaro Murata, and Hiroyuki Tsuhari. Development of Robots for Nuclear Power Plants and Their Application to New Fields. 57(4), 2020.

Robert J. Griffin, Georg Wiedebach, Sylvain Bertrand, Alexander Leonessa, and Jerry Pratt. Walking Stabilization Using Step Timing and Location Adjustment on the Humanoid Robot, Atlas. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 667–673, September 2017. doi: 10.1109/IROS.2017.8202223.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement Learning with Deep Energy-Based Policies, July 2017.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor, August 2018.

Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal Difference Learning for Model Predictive Control, July 2022.

Yandong Ji, Gabriel B. Margolis, and Pulkit Agrawal. DribbleBot: Dynamic Legged Manipulation in the Wild, April 2023.

Rob Knight, Pepijn Kooijmans, Thomas Wolf, Simon Alibert, Michel Aractingi, Dana Aubakirova, Adil Zouitine, Russi Martino, Steven Palma, Caroline Pascal, and Remi Cadene. Standard Open SO-100 & SO-101 Arms.

Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement Learning in Robotics: A Survey.

Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning Quadrupedal Locomotion over Challenging Terrain. *Science Robotics*, 5(47):eabc5986, October 2020. ISSN 2470-9476. doi: 10.1126/scirobotics.abc5986.

Seungjae Lee, Yibin Wang, Haritheja Etukuru, H. Jin Kim, Nur Muhammad Mahi Shafiullah, and Lerrel Pinto. Behavior Generation with Latent Actions, June 2024.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, July 2019.

Jianlan Luo, Charles Xu, Jeffrey Wu, and Sergey Levine. Precise and Dexterous Robotic Manipulation via Human-in-the-Loop Reinforcement Learning, October 2024.

Jianlan Luo, Zheyuan Hu, Charles Xu, You Liang Tan, Jacob Berg, Archit Sharma, Stefan Schaal, Chelsea Finn, Abhishek Gupta, and Sergey Levine. SERL: A Software Suite for Sample-Efficient Robotic Reinforcement Learning, March 2025.

Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 1 edition, May 2017. ISBN 978-1-316-66123-9 978-1-107-15630-2 978-1-316-60984-2. doi: 10.1017/9781316661239.

Gabriel B. Margolis, Ge Yang, Kartik Paigwar, Tao Chen, and Pulkit Agrawal. Rapid Locomotion via Reinforcement Learning, May 2022.

John McCormac, Ankur Handa, Andrew Davison, and Stefan Leutenegger. SemanticFusion: Dense 3D Semantic Mapping with Convolutional Neural Networks, September 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning, December 2013.

Lindsay Sanneman, Christopher Fourie, and Julie A. Shah. The State of Industrial Robotics: Emerging Technologies, Challenges, and Key Research Directions, October 2020.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust Region Policy Optimization, April 2017a.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, August 2017b.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, Simon Alibert, Matthieu Cord, Thomas Wolf, and Remi Cadene. SmolVLA: A Vision-Language-Action Model for Affordable and Efficient Robotics, June 2025.

Bruno Siciliano and Oussama Khatib, editors. *Springer Handbook of Robotics*. Springer Handbooks. Springer International Publishing, Cham, 2016. ISBN 978-3-319-32550-7 978-3-319-32552-1. doi: 10.1007/978-3-319-32552-1.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic Policy Gradient Algorithms. In *Proceedings of the 31st International Conference on Machine Learning*, pages 387–395. PMLR, January 2014.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning Series. The MIT Press, Cambridge, Massachusetts, second edition edition, 2018. ISBN 978-0-262-03924-6.

Chen Tang, Ben AbbateMatteo, Jiaheng Hu, Rohan Chandra, Roberto Martín-Martín, and Peter Stone. Deep Reinforcement Learning for Robotics: A Survey of Real-World Successes, September 2024.

Yang Tang, Chaoqiang Zhao, Jianrui Wang, Chongzhen Zhang, Qiyu Sun, Weixing Zheng, Wenli Du, Feng Qian, and Juergen Kurths. Perception and Navigation in Autonomous Systems in the Era of Learning: A Survey. *IEEE Transactions on Neural Networks and Learning Systems*, 34(12):9604–9624, December 2023. ISSN 2162-237X, 2162-2388. doi: 10.1109/TNNLS.2022.3167688.

Russ Tedrake. Robotic Manipulation. Perception, Planning and Control., a.

Russ Tedrake. Underactuated Robotics. Algorithms for Walking, Running, Swimming, Flying, and Manipulation, b.

Gabriele Tiboni, Karol Arndt, and Ville Kyrki. DROPO: Sim-to-Real Transfer with Offline Domain Randomization, January 2023.

Gabriele Tiboni, Pascal Klink, Jan Peters, Tatiana Tommasi, Carlo D’Eramo, and Georgia Chalvatzaki. Domain Randomization via Entropy Maximization, March 2024.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World, March 2017.

Chong Zhang, Wenli Xiao, Tairan He, and Guanya Shi. WoCoCo: Learning Whole-Body Humanoid Control with Sequential Contacts, November 2024.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware, April 2023.