Robotic reinforcement learning

Arthur Louette (arthur.louette@uliege.be)

April 15, 2025

Outline

Introduction

Definitions

What is a robot?

Definition (Robot)

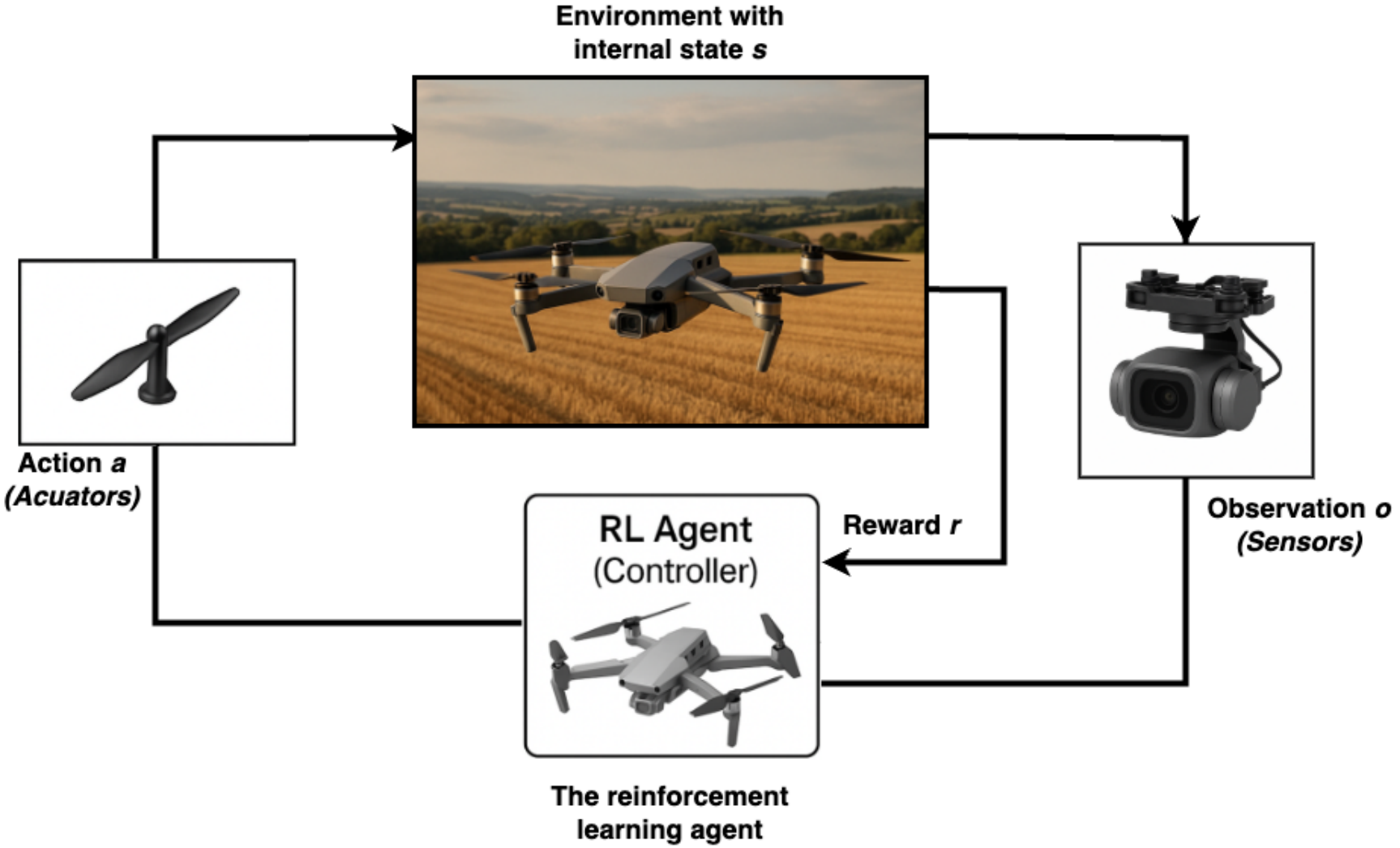
A robot is a goal oriented machine that can sense, plan, and act [Corke, 2017].

This lecture will focus on reinforcement learning to control robots.

Definition (Robotic agent)

A robotic agent is an autonomous agent that perceives its environment via sensors, makes decisions using learned controller, and executes actions through actuators to achieve specific goals. It learns and adapts its behavior over time through interaction with the environment.

Robotic agent interaction loop



POMDP

A robotic agent only perceives partial information about its state through sensors.

This lecture formalizes the problems with a single robotic agent that has to solve one task in the real world as POMDPs.

A POMDP is represented by its model $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \mathcal{O}, T, R, O, P, \gamma)$:

- States $s_t \in \mathcal{S}$,
- Actions $a_t \in \mathcal{A}$,
- Observations $o_t \in \mathcal{O}$,
- Transition distribution $T(s_{t+1}|s_t, a_t)$,
- Reward function $r_t = R(s_t, a_t)$,
- Observation distribution $O(o_t|s_t)$,
- Initial distribution $P(s_0)$,
- Discount factor $\gamma \in [0, 1[$.

Challenges for solving POMDPs in the real world

Challenges for solving POMDPs in the real world:

- Safety: The environment exploration with random actions could break the robot. Learning safe policies that will not break the robot at execution time is crucial.

Challenges for solving POMDPs in the real world

Challenges for solving POMDPs in the real world:

- Safety: The environment exploration with random actions could break the robot. Learning safe policies that will not break the robot at execution time is crucial.
- Sample efficiency: As you operate in the real world, the training time is inherently limited by the speed of real-time operations, often resulting in extended learning.

Challenges for solving POMDPs in the real world

Challenges for solving POMDPs in the real world:

- Safety: The environment exploration with random actions could break the robot. Learning safe policies that will not break the robot at execution time is crucial.
- Sample efficiency: As you operate in the real world, the training time is inherently limited by the speed of real-time operations, often resulting in extended learning.
- Real-time decision: the robotic agent is constrained to swiftly process sensory information, compute actions, and execute them within the time constraints imposed by the dynamic of the real-world environment.

Challenges for solving POMDPs in the real world

Challenges for solving POMDPs in the real world:

- Safety: The environment exploration with random actions could break the robot. Learning safe policies that will not break the robot at execution time is crucial.
- Sample efficiency: As you operate in the real world, the training time is inherently limited by the speed of real-time operations, often resulting in extended learning.
- Real-time decision: the robotic agent is constrained to swiftly process sensory information, compute actions, and execute them within the time constraints imposed by the dynamic of the real-world environment.
- Diversity of the deployed conditions: Real-world environments are inherently unpredictable, with factors such as lighting conditions, surface textures, and object placements varying unpredictably, posing challenges for generalization.

Challenges for solving POMDPs in the real world

Challenges for solving POMDPs in the real world:

- Safety: The environment exploration with random actions could break the robot. Learning safe policies that will not break the robot at execution time is crucial.
- Sample efficiency: As you operate in the real world, the training time is inherently limited by the speed of real-time operations, often resulting in extended learning.
- Real-time decision: the robotic agent is constrained to swiftly process sensory information, compute actions, and execute them within the time constraints imposed by the dynamic of the real-world environment.
- Diversity of the deployed conditions: Real-world environments are inherently unpredictable, with factors such as lighting conditions, surface textures, and object placements varying unpredictably, posing challenges for generalization.
- Sensor noise and uncertainty: Sensors used by robots to perceive their surroundings can be noisy and prone to inaccuracies, leading to uncertainty in the observations and complicating learning algorithms.

Single Robotic Agent POMDP

In this lecture, we will consider two approaches for solving single robotic agent POMDP:

- Learning in simulation
- Learning in the real world

Learning in simulation

Motivations

Learning in simulation has several advantages:

- Safety: If the real environment can be simulated, policies can be learned safely without the risk of damaging the robot.
- Increased data acquisition: Less sample efficiency is required as more samples can be collected for the same time budget, thanks to the parallelization and speed of the simulator, especially if it runs on GPUs.
- Realism: Advanced simulation techniques can replicate real-world variability, including lighting conditions, textures, and object physics.
- Ease to reset: Simulation is easier to reset than hardware, where manual reset or replacing the robot is needed.

Robotic simulator

In robotics, a robotic simulator uses a physics engine to replicate real-world environments and enables learning in simulation and subsequent transfer of learned policies to reality.

Definition (Physics Simulator)

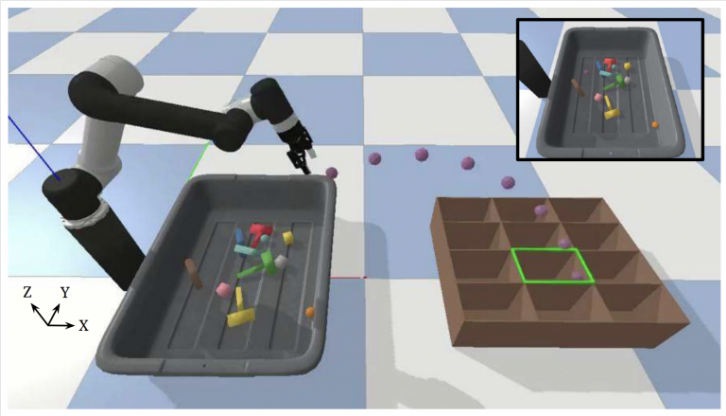
A physics simulator is a software tool that replicates the behavior of real-world physics within a virtual environment. It enables the simulation of interactions between objects, allowing for realistic training scenarios for robotic tasks.

Definition (Robotic Simulator)

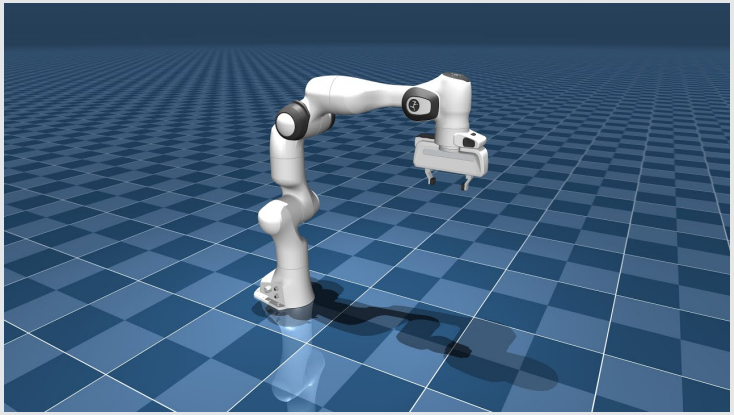
A robotic simulator is a software that emulates robotic systems, featuring a physics simulator for realistic modelling, collision detection, friction models, a user and a graphical user interface, scene and mesh import, API support (e.g., C++/Python), and a library of joints, actuators, and sensors for easy simulation setup.

Some robotic simulator

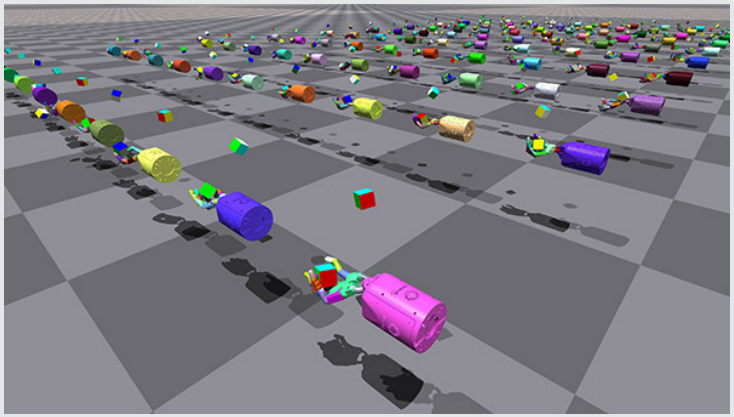
PyBullet



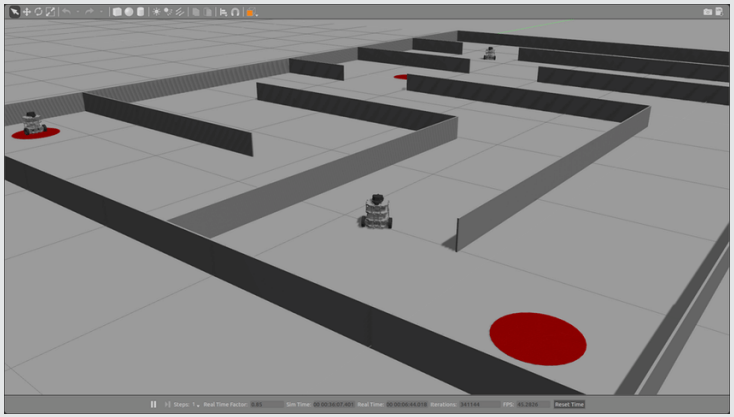
MuJoCo



Isaac Sim



Gazebo



Among others: AirSim, CoppeliaSim, CARLA, SOFA, Webots ...

Definition (Robotic simulator)

Let us denote the set of possible POMDP as $\mathcal{U}$. A robotic simulator is modeled as a distribution of parametrized POMDP $\Psi :\in \Delta(\mathcal{U})$. Each POMDP is represented by a model $\mathcal{P}_\psi = (\mathcal{S}, \mathcal{A}, \mathcal{O}, T_\psi, R_\psi, O_\psi, P_\psi, \gamma)$ with $\mathcal{P}_\psi \sim \Psi(.)$.

In this lecture, the real world is modeled as a distribution of parametrized POMDP $\Psi^* \in \Delta(\mathcal{U})$.

Definition (Real world)

Let us denote the set of possible POMDP as $\mathcal{U}$. The real world is modeled as a distribution of parametrized POMDP $\Psi^* \in \Delta(\mathcal{U})$. Each POMDP is represented by a model $\mathcal{P}_{\psi^*}$ with unknown dynamics T_{ψ^*}, unknown observation distribution O_{ψ^*}, unknown reward function R_{ψ^*} and unknown initial state distribution P_{ψ^*} due to modeling inaccuracies and sensor differences.

Return of a policy for a distribution of POMDPs

Let $h_t = (o_0, a_0, \ldots, a_{t-1}, o_t) \in \mathcal{H}$ denote the history at time step t.

Definition (Return of a policy for a distribution of POMDPs)

The return of a policy $\eta \in H = \mathcal{H} \to \mathcal{A}$ for a distribution of POMDPs $\Psi \in \Delta(\mathcal{U})$ is defined as:

$$J_\Psi(\eta) = \mathop{\mathbb{E}}_{\mathcal{P}_\psi \sim \Psi(.)} \left[\mathop{\mathbb{E}}_{\substack{s_0 \sim P_\psi(\cdot) \\ a_t \sim \eta(\cdot|h_t) \\ s_{t+1} \sim T_\psi(\cdot|s_t, a_t) \\ o_{t+1} \sim O_\psi(\cdot|s_t)}} \left[\sum_{t=0}^{\infty} \gamma^t R_\psi(s_t, a_t) \right] \right]$$

Simulation-to-Reality Gap

As the real world POMDP distribution Ψ^* is unknown, the simulator Ψ generates data to train a history-dependent stochastic policy $\eta_\Psi \in H = \mathcal{H} \to \mathcal{A}$ such that it minimizes the simulation-to-reality gap.

Definition (Simulation-to-Reality Gap)

The Simulation-to-Reality Gap is defined as the difference between the return of a policy η_Ψ learned in the simulator Ψ and the optimal policy η^* in the real-world Ψ^*.

The Simulation-to-Reality Gap is mathematically defined as:

$$Gap(\eta_\Psi) = J_{\Psi^*}(\eta^*) - J_{\Psi^*}(\eta_\Psi)$$

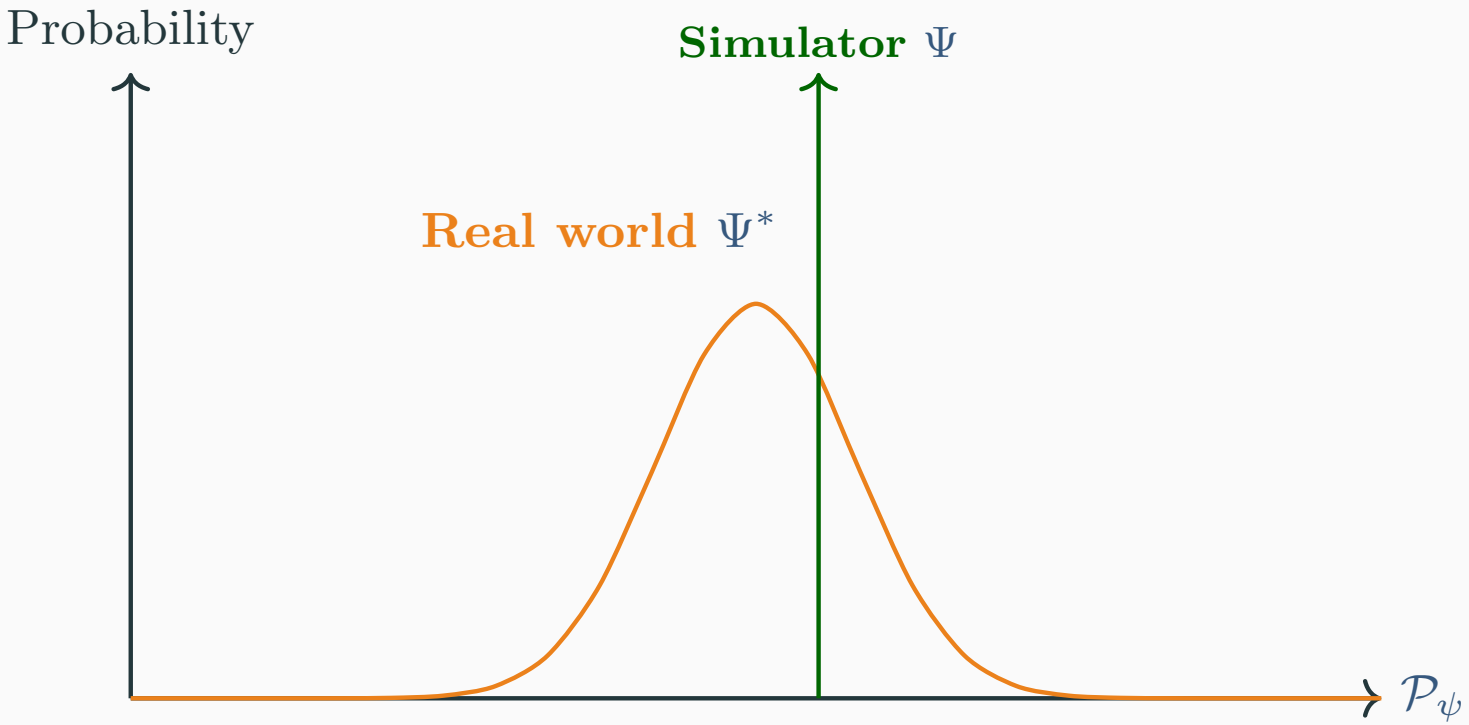
where $\eta^* = argmax_{\eta \in H} J_{\Psi^*}(\eta)$

System identification

System identification studies how to learn a model of the system dynamics.

It is an important step to reduce the simulation-to-reality gap.
Example: In drone racing, it implies to identify the drag of the drone, the position of the gates, ...

1. Estimate Ψ^* from the real world.
2. Sample from Ψ to train.



System identification

Problem:

- It is extremely difficult to estimate all the parameters precisely.
- It is even more difficult to identify them all. Think about friction forces, battery behavior, wind disturbances, motor delays...

In practice, we often rely on partial identification and design policies that are robust to the remaining uncertainty.

Another approach is to learn a distribution over possible parameters, instead of a single set.
This enables domain randomization: sampling many plausible dynamics during training to improve generalization.

Yet, the better the identification, the smaller the sim-to-real gap.

Drone Racing Example

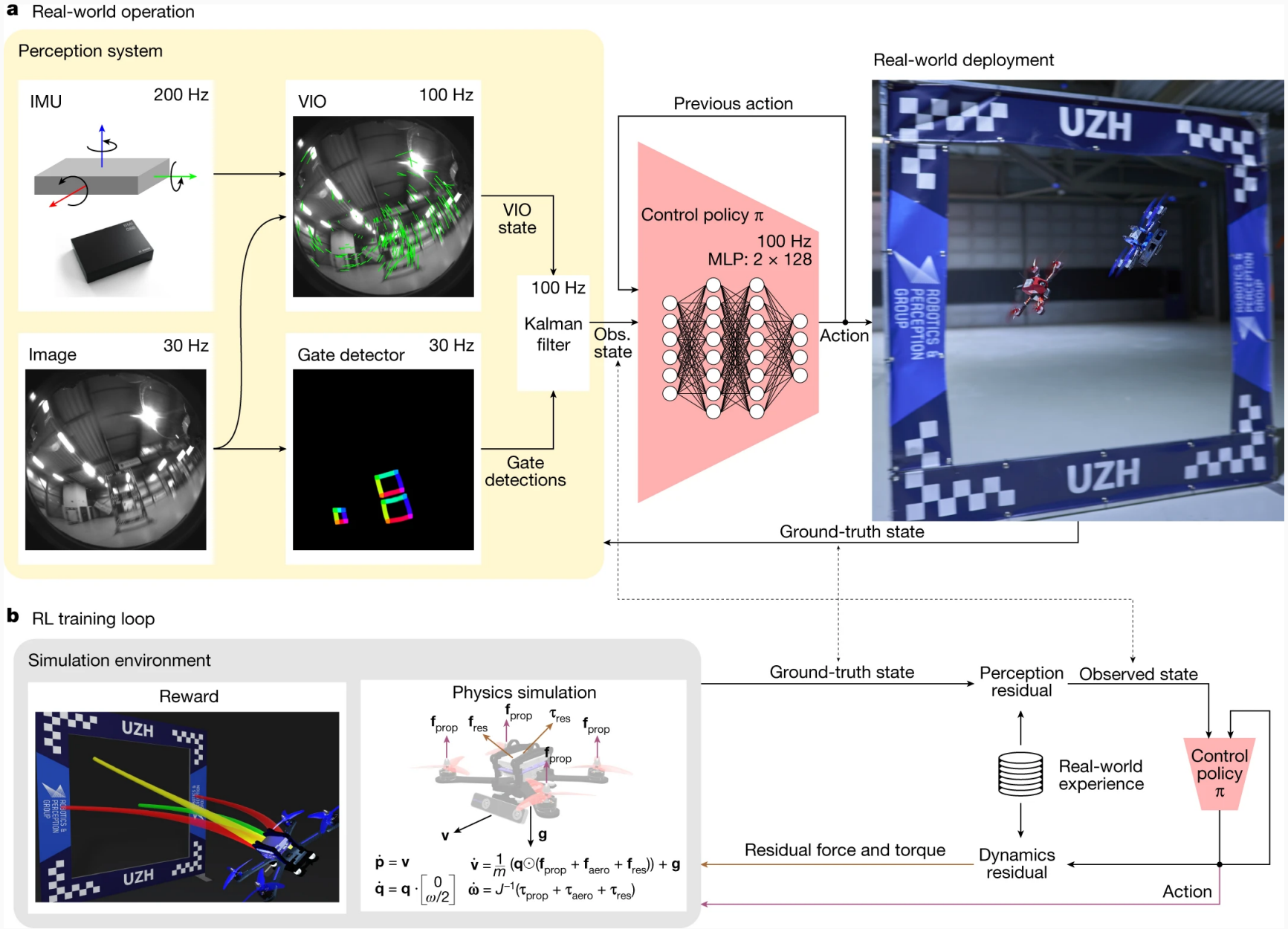


Figure from Kaufmann et al. [2023]

Domain randomization

Definition (Domain Randomization)

Let $\mathcal{U}$ be a set of POMDP, domain randomization consist in finding a distribution $\Psi \in \Delta(\mathcal{U})$ such that the optimal policy derived from Ψ i.e. $\eta^*_\Psi = argmax_{\eta \in H} J_\Psi(\eta)$ minimizes the simulation to reality gap:

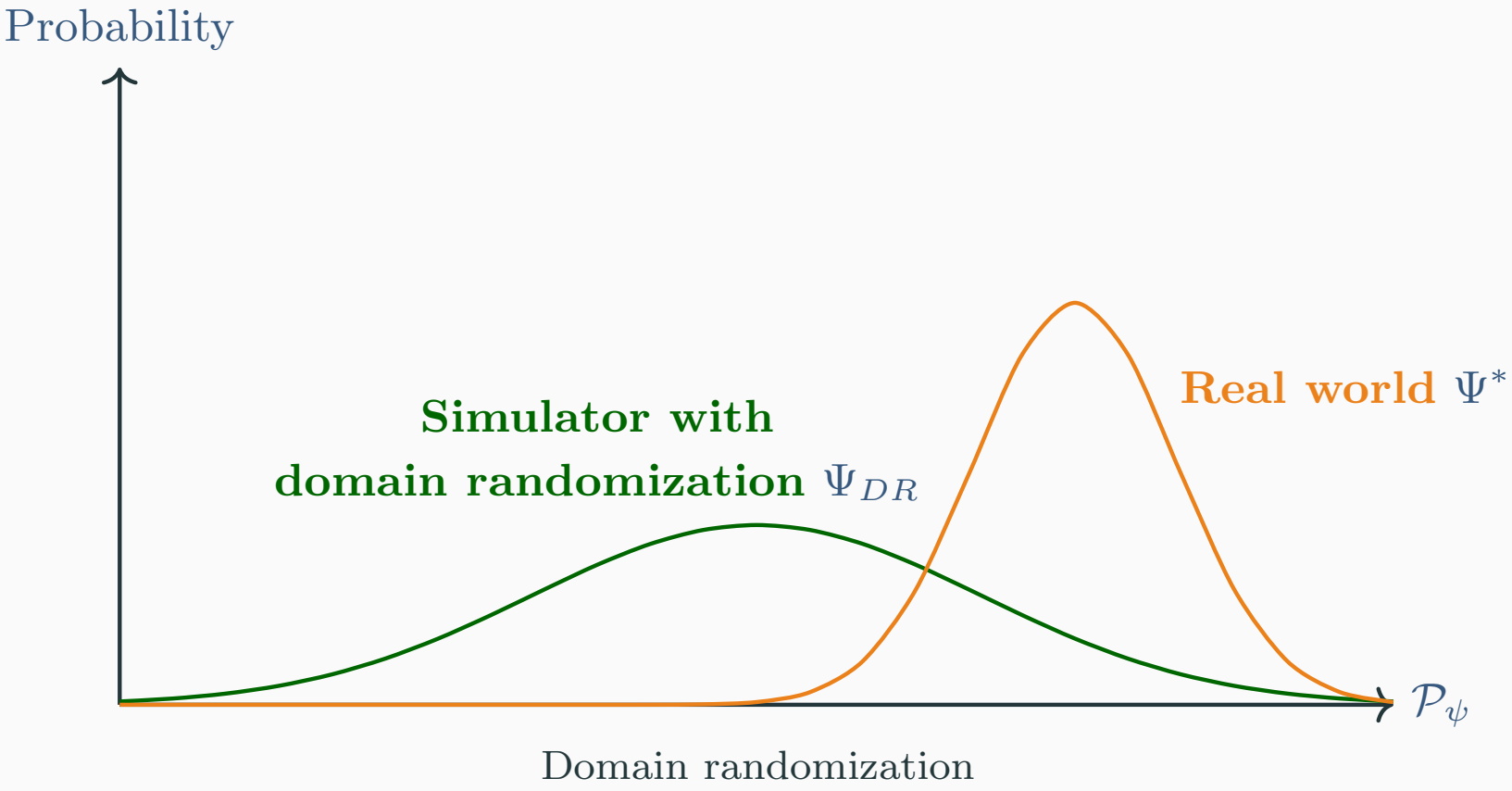
$$\Psi_{DR} = \underset{\Psi \in \Delta(\mathcal{U})}{argmin}\, Gap(\eta^*_\Psi)$$

.

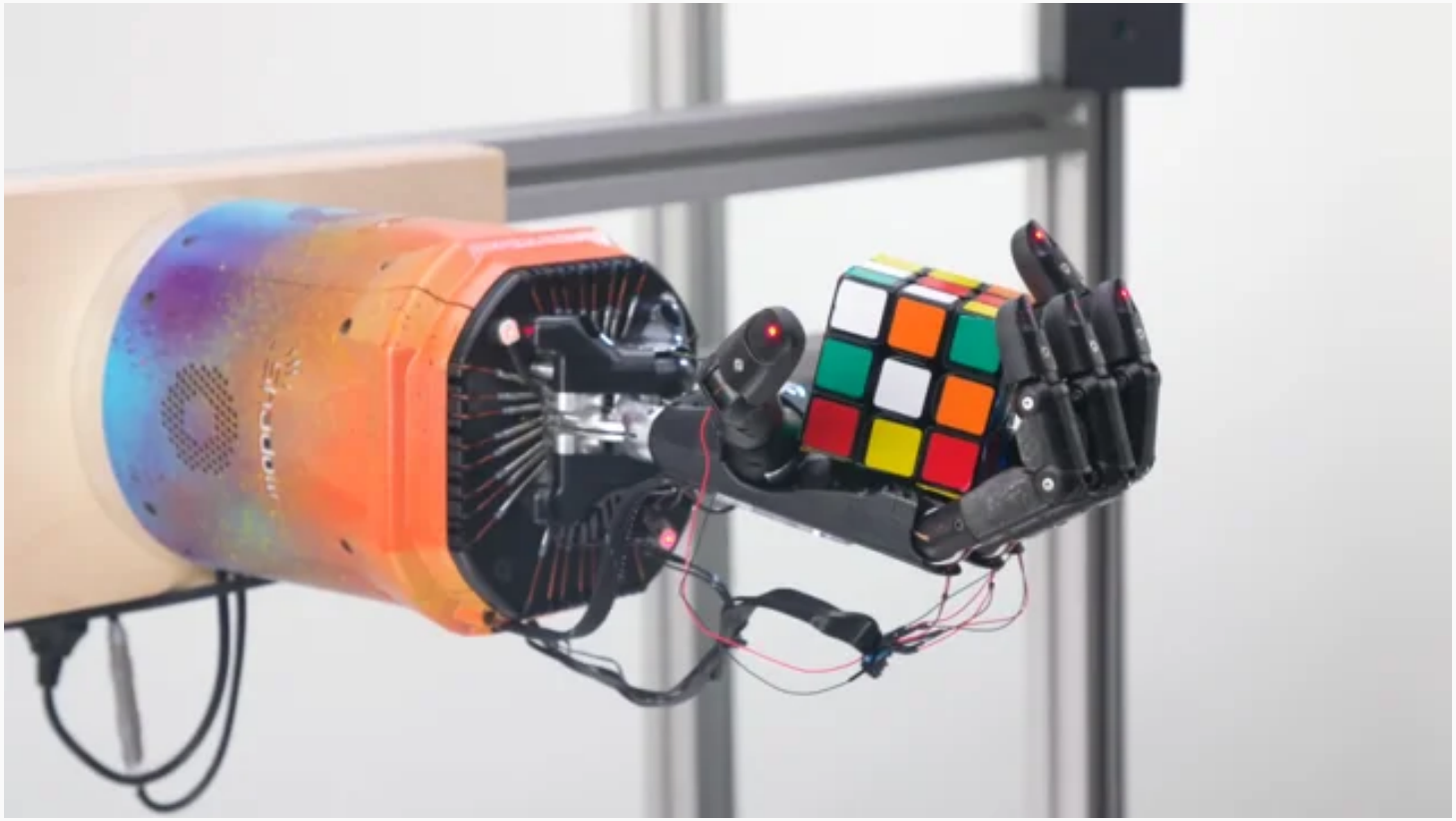
Obviously, Ψ^* is a solution of this optimization problem but it is often unknown.

In practice, the distribution Ψ is often defined with a uniform or gaussian distribution over the unknown parameters of the system dynamics but more sophisticated techniques exist.

The hope is to generalize to the real world at the cost of less sample efficiency due to the training in a distribution of POMDPs instead of a single POMDP.



Domain randomization: example



https://openai.com/research/solving-rubiks-cube

Curriculum Learning

Idea: Use a curriculum of task difficulty to learn complex policies.

Definition (Curriculum Learning)

Let Ψ be a distribution of POMDPs. A curriculum is defined as a sequence of environment distributions

$$\mathcal{C} = \{\Psi_0, \Psi_1, \ldots, \Psi_N\}, \quad \Psi_n \in \Delta(\mathcal{U}) \text{ for } n = 0, 1, \ldots, N,$$

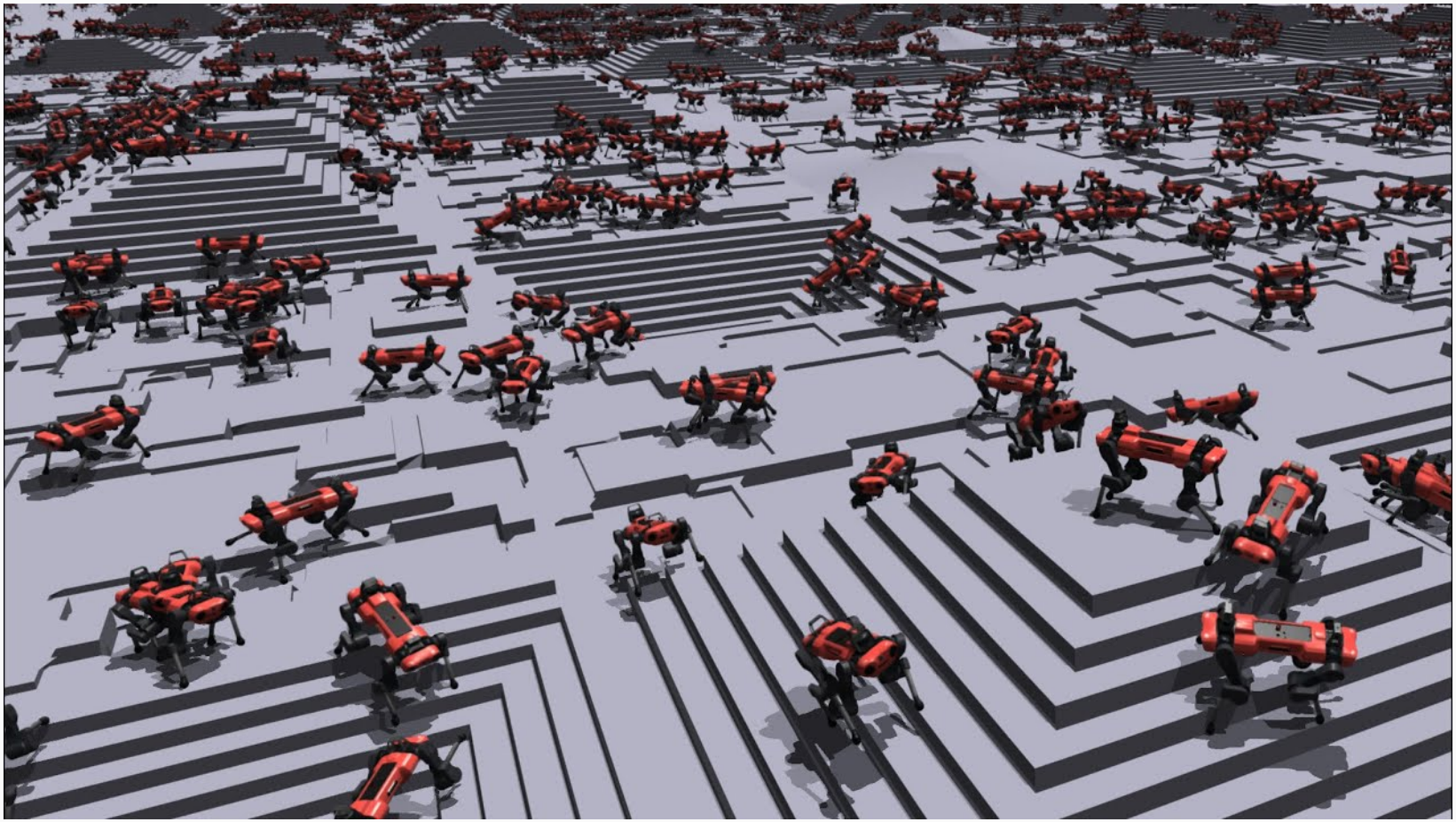
The objective of curriculum learning is to obtain a final policy

$$\eta_N^* = \arg\max_{\eta \in H} J_{\Psi_N}(\eta),$$

which minimizes the simulation-to-reality gap.

Ψ_0 corresponds to an *easy* version of the task (e.g., with simplified dynamics or lower difficulty).
Ψ_N approximates the target distribution, typically the real-world distribution denoted by Ψ^*.

Curriculum learning Example



Learning to Walk in Minutes Using Massively Parallel Deep RL [Rudin et al., 2022].

Data collection and policy update

Current on-policy reinforcement learning algorithms are divided into two parts: data collection and policy update.

- Data collection: Each step consists of policy inference, simulation, reward, and observation calculation.
- Policy update: The policy update, which corresponds to back-propagation for neural networks, is easily performed in parallel on the GPU.

Data collection and policy update

Current on-policy reinforcement learning algorithms are divided into two parts: data collection and policy update.

- Data collection: Each step consists of policy inference, simulation, reward, and observation calculation.
- Policy update: The policy update, which corresponds to back-propagation for neural networks, is easily performed in parallel on the GPU.

However, in many current pipelines, simulation and reward/observation computations run on the CPU, while policy inference and updates run on the GPU.

This split introduces a communication bottleneck over PCIe, where data transfer can be up to 50× slower than GPU compute time [Gregg and Hazelwood, 2011].

Overcoming bottlenecks with parallel simulation

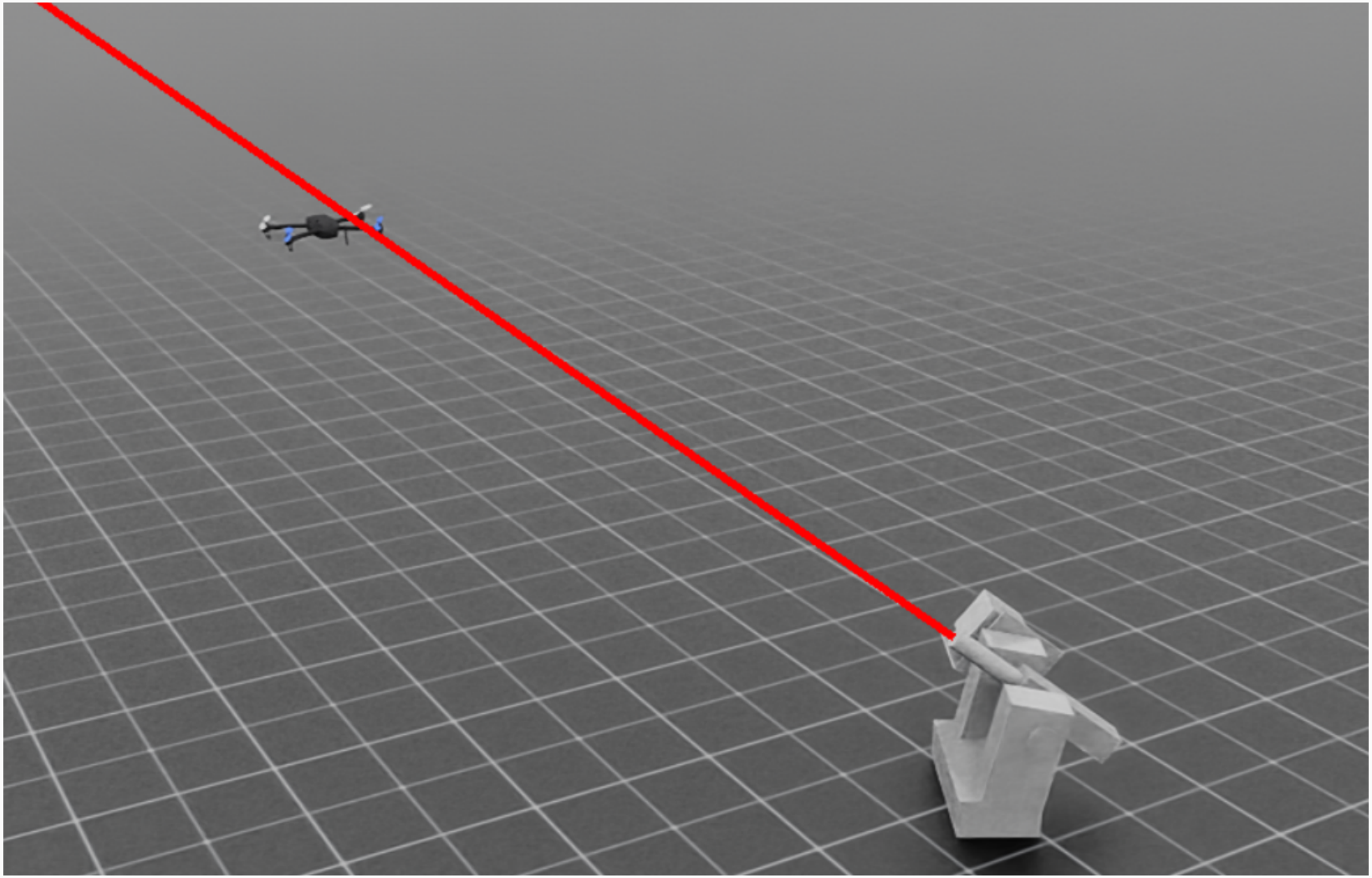
CPU-based simulation offers only limited parallelism through multi-core processing, which quickly becomes constrained by core count and memory limits.

Move both data collection and policy update entirely to the GPU.

Examples of GPU/TPU-accelerated simulators:

- Isaac Sim (NVIDIA): Robotic simulation on the GPU.
- Brax (Google): Physics simulation on TPUs and GPUs written in Jax.

Application: CUAS with a weapon station.



CUAS with a weapon station.

Limitations of learning in simulation

The algorithms we have seen until now solve some challenges for robotic reinforcement learning.

Learning in simulation:

- Safety: Learning in simulation avoid to break robots during the learning phase.
- Real-time decision: RL has the advantage of shifting computational complexity to the learning phase in opposition to optimal control.

Limitations of learning in simulation

The algorithms we have seen until now solve some challenges for robotic reinforcement learning.

Learning in simulation:

- Safety: Learning in simulation avoid to break robots during the learning phase.
- Real-time decision: RL has the advantage of shifting computational complexity to the learning phase in opposition to optimal control.
- Sample efficiency: The need of sample efficiency algorithms is avoided with GPU for data collection and learning.
- Diversity of the deployed conditions: The real world is approximated with a distribution of POMDPs using system identification, domain randomization and curriculum.
- Sensor noise and uncertainty: The sensors and the noise is modeled by system identification and domain randomization. However, some sensors are hard to replicate in simulation.

Learning in the real-world

Learning in the real-world:

- Diversity of the deployed conditions: There is no problem to overfit the real world by learning directly in the target domain.
- Sensor noise and uncertainty: Learning directly with the real sensor avoids the problem of simulation-to-reality gap.

Learning in the real-world:

- Diversity of the deployed conditions: There is no problem to overfit the real world by learning directly in the target domain.
- Sensor noise and uncertainty: Learning directly with the real sensor avoids the problem of simulation-to-reality gap.
- Real-time decision: How to learn online while handling the execution time constraint?
- Safety: How to reset and avoid catastrophic failure during the learning phase?
- Sample efficiency: How can we learn with less samples as we cannot collect as many steps than in simulation? (e.g. learning to walk: 1.4×10^7 samples, drone racing: 10^8 samples)

Online real-world learning: Safety and real-time learning

We will explore online real-world learning for quadrupedal locomotion.
In this section, we will consider an infinite time horizon MDP
$\mathcal{M} = \{\mathcal{S}, \mathcal{A}, T, R, P, \gamma\}$.

In the context of locomotion for quadrupeds, ensuring safety is straightforward:

- A non-recoverable failure is unlikely, as the robot can only fall.
- Resetting the environment simply involves repositioning the quadruped on its feet.

Online real-world learning: Safety and real-time learning

We will explore online real-world learning for quadrupedal locomotion.
In this section, we will consider an infinite time horizon MDP
$\mathcal{M} = \{\mathcal{S}, \mathcal{A}, T, R, P, \gamma\}$.

In the context of locomotion for quadrupeds, ensuring safety is straightforward:

- A non-recoverable failure is unlikely, as the robot can only fall.
- Resetting the environment simply involves repositioning the quadruped on its feet.

A naive implementation can be slower than data collection to learn in real-time:

- Asynchronous training and trial-based updates: Delays between interaction and learning.
- Synchronous training: Need fast implementation, in Jax for instance.

Online real-world learning: Sample efficiency

In simulation, we have considered online-policy model-free algorithms such as PPO because collecting samples is cheap.

In the real world, we will consider off-policy model-free algorithms because you can recycle old samples via a replay buffer to learn the optimal policy.

More precisely, we will focus on off-policy actor-critic algorithms such as Soft Actor-Critic and its variants.[Haarnoja et al., 2019]

Off-policy actor-critic algorithms

Actor-critic algorithms interleave between critic and actor updates.

Critic Update:

$$J(\phi) = \mathbb{E}_{(s,a,s')\sim\mathcal{D}}\left[(Q_\phi(s,a) - y)^2\right]$$

$$y = R(s,a) + \gamma Q_{\phi'}(s', a'), \quad a' \sim \pi_\theta(\cdot|s')$$

ϕ' is an exponentially averaged target network.
$\mathcal{D}$ is an experience replay buffer.

Actor Update:

$$J(\pi) = \mathbb{E}_{s\sim\mathcal{D}, a\sim\pi_\theta(\cdot|s)}\left[Q_\phi(s,a)\right]$$

Soft Actor-Critic (SAC)

Algorithm 0: Soft Actor-Critic

Initialize Q-function parameters ϕ_1, ϕ_2, policy parameters θ
Initialize target networks $\hat{\phi}_1 \leftarrow \phi_1$, $\hat{\phi}_2 \leftarrow \phi_2$
Initialize empty replay buffer $\mathcal{D} \leftarrow \emptyset$
for *each iteration* **do**
  for *each environment step* **do**
    $\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
    $\mathbf{s}_{t+1} \sim T(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, R(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
  for *each gradient step* **do**
    for $i \in \{1, 2\}$ **do**
      $\phi_i \leftarrow \phi_i - \lambda_Q \nabla_{\phi_i} J_Q(\phi_i)$
    $\theta \leftarrow \theta - \lambda_\pi \nabla_\theta J_\pi(\theta)$
    $\alpha \leftarrow \alpha - \lambda_\alpha \nabla_\alpha J(\alpha)$
    for $i \in \{1, 2\}$ **do**
      $\hat{\phi}_i \leftarrow \tau\phi_i + (1 - \tau)\hat{\phi}_i$
Output: ϕ_1, ϕ_2, θ

See Haarnoja et al. [2019] for more details.

Policy Objective (θ):

$$J_\pi(\theta) = \mathbb{E}_{s_t, \varepsilon_t} \left[\alpha \log \pi_\theta(a_t|s_t) - Q_\phi(s_t, a_t) \right]$$

Q-Function Objectives (ϕ_1, ϕ_2):

$$J_Q(\phi_i) = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1})} \left[\frac{1}{2} \left(Q_{\phi_i}(s_t, a_t) - y_t \right)^2 \right]$$

$$y_t = r_t + \gamma \left(\min_j Q_{\hat{\phi}_j}(s_{t+1}, a_{t+1}) - \alpha \log \pi_\theta(a_{t+1}|s_{t+1}) \right)$$

Temperature Tuning (α):

$$J(\alpha) = \mathbb{E}_{a_t \sim \pi_\theta} \left[-\alpha \log \pi_\theta(a_t|s_t) - \alpha \bar{\mathcal{H}} \right]$$

See Haarnoja et al. [2019] for more details.

Update-to-data ratio

Off-policy algorithms allow us to decouple data collection from training.
Therefore, we can reuse this data to update the critic and actor multiple times per environment step.
This leads to a higher update-to-data ratio (UTD).

Definition (update-to-data ratio (UTD))

The update-to-data ratio (UTD) is defined as the ratio between the number of times we update a network and the number of data collected.

Increasing the UTD data improves the sample efficiency.
However, the more we reuse the same data, the higher the risk that the critic will overfit to that data.

Regularization

Definition (Regularization techniques)

Regularization techniques are a family of methods that reduce the generalization gap between training and test performance. [Prince, 2023]

Regularization techniques are essential in controlling overfitting and ensuring stability, particularly when training with high UTD ratios.

Examples of regularization methods:

- **Dropout** – Randomly drops activations during training.
- **Ensembling** – Combines multiple independent estimates to reduce variance.
- **Layer Normalization** – Normalizes layer inputs to stabilize training.

These techniques help ensure that repeated updates do not lead to a critic overfitting its training data.

Dropout

Definition (Dropout)

Dropout randomly sets a percentage of the hidden units of a neural network (typically 50%) to zero during each SGD iteration.

This reduces reliance on individual hidden units and encourages smaller weights, making the network more stable.

Ensembling of Critic Networks

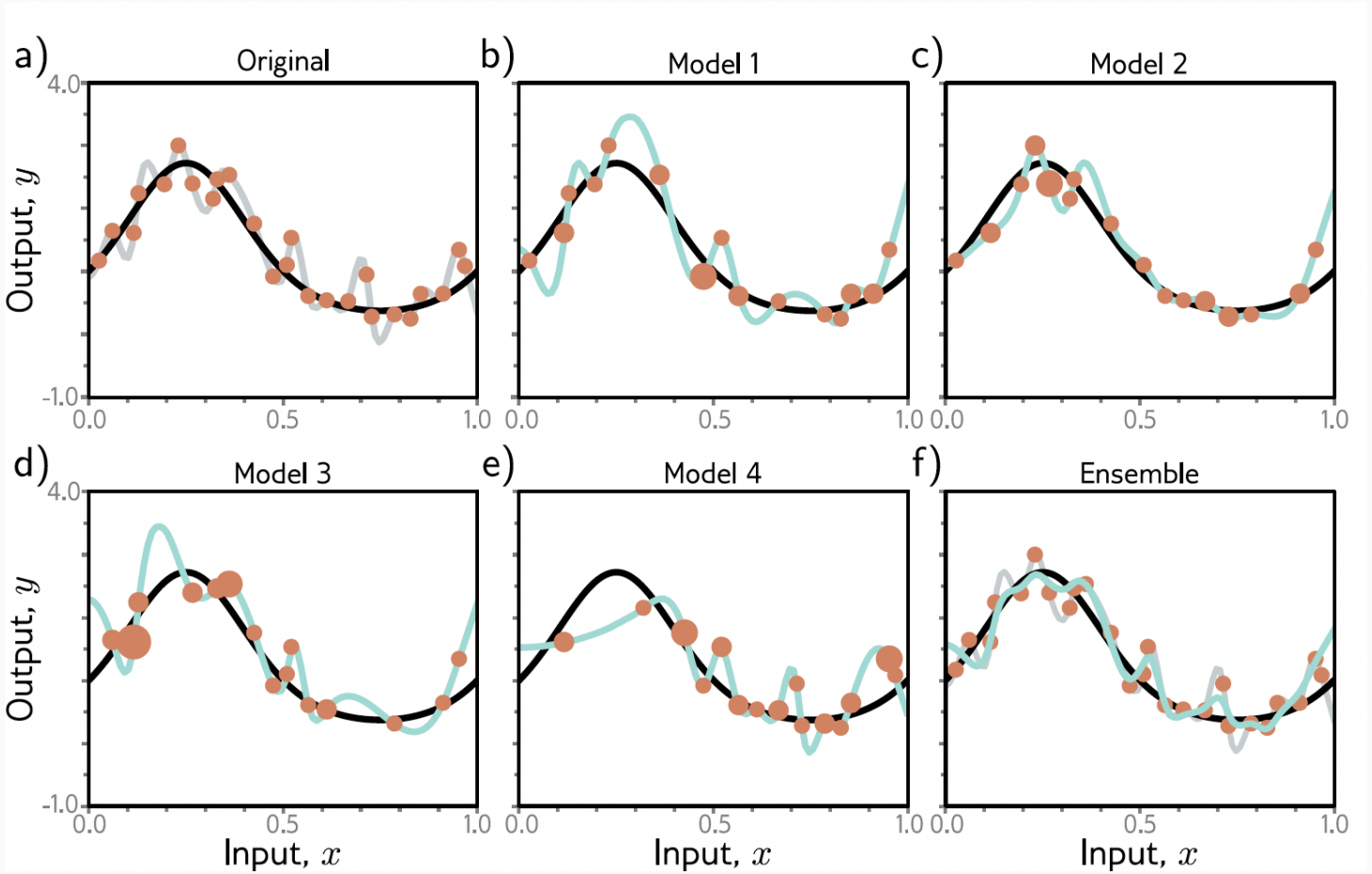
Definition (Ensembling)

Ensembling consists of training multiple independent estimator and averaging their predictions.

In our case, these estimators are multiple independent critics. A group of N critic networks, each parameterized by θ_i estimating the value function $Q_{\theta_i}(s, a)$ is called an ensemble. These critics can be combined by taking the mean of the output for instance:

$$Q_{\text{ensemble}}(s, a) = \frac{1}{N} \sum_{i=1}^{N} Q_{\theta_i}(s, a).$$

Ensembling example



Ensemble methods. a) Fitting a single model (gray curve) to the entire dataset (orange points). b–e) Four models created by re-sampling the data with replacement (bagging) four times (size of orange point indicates number of times the data point was re-sampled). f) When we average the predictions of this ensemble, the result (cyan curve) is smoother than the result from panel (a) for the full dataset (gray curve) and will probably generalize better. Example from Prince [2023].

Layer Normalization

Definition (Layer Normalization)

Layer normalization is a technique that normalizes the inputs to a layer for each training example, ensuring the activations have zero mean and unit variance.

For an input vector $x \in \mathbb{R}^d$:

$$\mu = \frac{1}{d}\sum_{i=1}^{d} x_i, \quad \sigma^2 = \frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2$$

The normalized output is computed as:

$$\text{LN}(x) = \gamma \frac{x - \mu}{\sqrt{\sigma^2 + \varepsilon}} + \beta$$

Where:

- γ and β are learnable scale and shift parameters,
- ε is a small constant for numerical stability.

This normalization ensures each layer's inputs have a consistent distribution, promoting stability even with many gradient updates [Ba et al., 2016].

SAC variants

SAC variants:

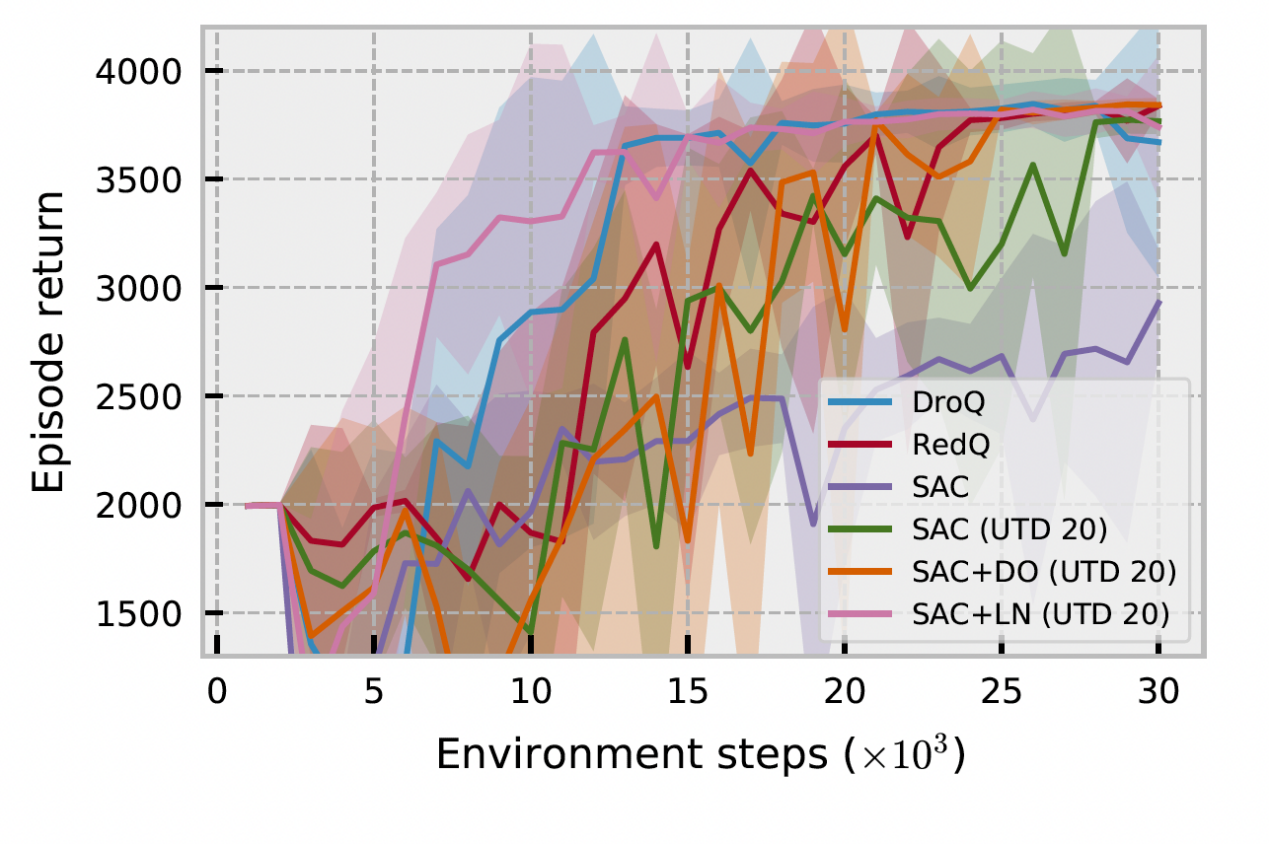
- DroQ: regularizes the critic networks with dropout and layer normalization. [Hiraoka et al., 2022]
- RedQ: extends SAC with a large ensemble of critics and computes target values by minimizing over a random subset of them. [Chen et al., 2021]
- SAC (20 UTD): Soft Actor-Critic with increased number of critic updates made per time-step.
- SAC + DO: Soft Actor-Critic with dropout
- SAC + LN: Soft Actor-Critic with layer normalization

Locomotion task in real world

https://www.youtube.com/watch?v=YO1USfn6sHY

Laura Smith, Ilya Kostrikov, Sergey Levine. A Walk in the Park: Learning to Walk in 20 Minutes With Model-Free Reinforcement Learning. 2023

Regularization and higher update-to-data ratio



Regularization (ensemble methods, dropout, layer normalization...) and a higher update-to-data ratio increases the sample efficiency.

Laura Smith, Ilya Kostrikov, Sergey Levine. A Walk in the Park: Learning to Walk in 20 Minutes With Model-Free Reinforcement Learning. 2023

Limitations of real world learning

Learning in the real world overcomes some limitations. Yet, some challenges remain open:

- Safety: For some task, it is difficult to ensure safety and reset.
- Real-time decision: When dealing with larger input space such as images, the inference and learning time can increase.
- Sample efficiency: Training from scratch is still a challenge for some tasks.

Idea: Can we leverage prior data to learn faster?

Generalist robot policies

Generalist models

In various fields like natural language processing and computer vision, there is a trend toward using generalized pretrained models as a foundation for different tasks. These models are then fine-tune for a particular application. The success of these models rely on large datasets.

Questions:

- Is it possible to build such in robotics?
- Does the generalist model beat the specialist in robotics?

Recently, the community has put efforts to create large datasets of variable robots in different tasks [Collaboration et al., 2024].

Foundation models

Definition (Foundation models)

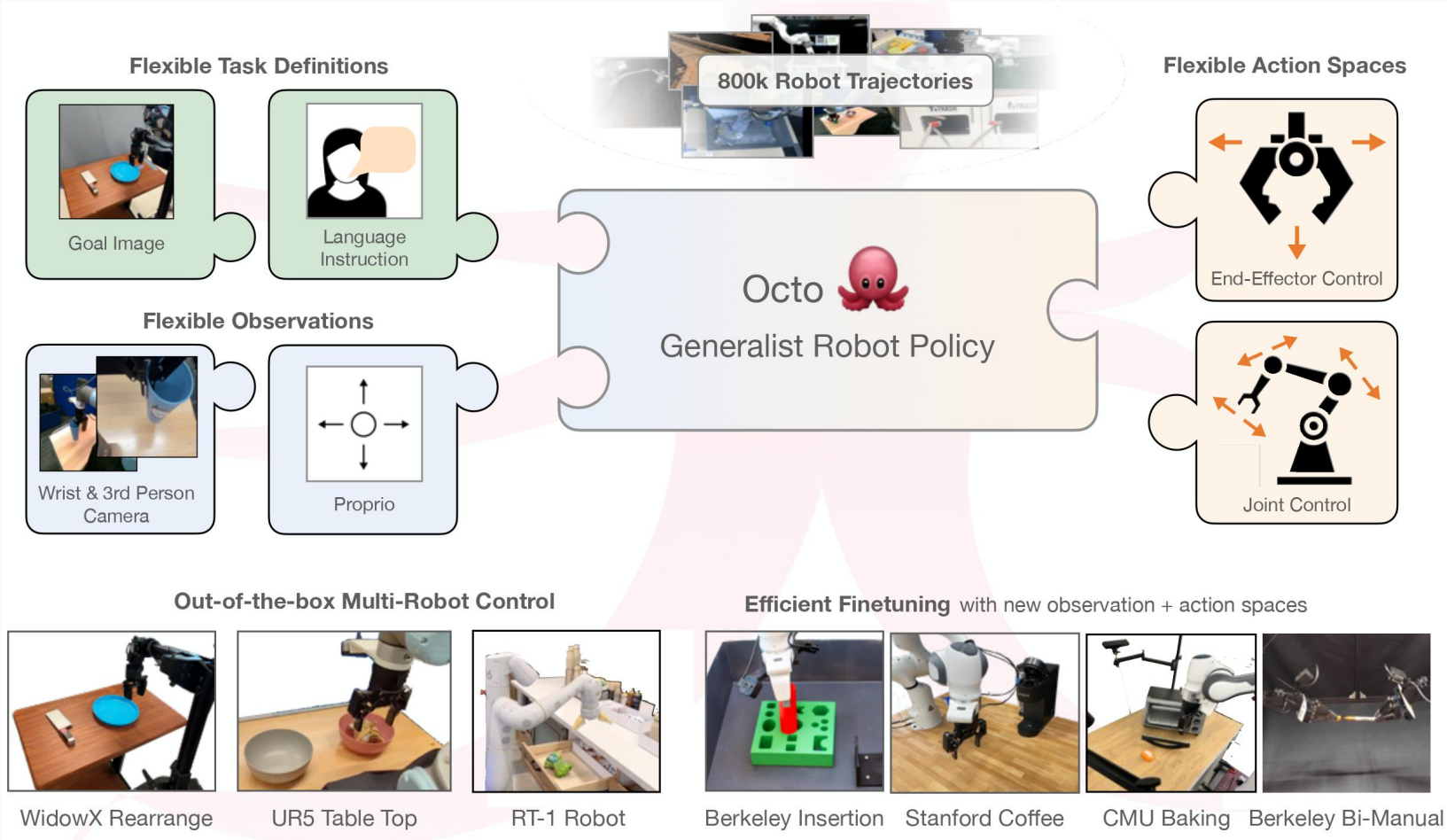
A foundation model is a machine learning model that is trained on broad data such that it can be applied across a wide range of use cases.

Traditional deep reinforcement learning models in robotics are trained on small datasets tailored for specific tasks, which limits their adaptability across diverse applications.

In contrast, foundation models pretrained on internet-scale data appear to have superior generalization capabilities, and in some instances display an emergent ability to find zero-shot solutions to problems that are not present in the training data.

Reinforcement learning is can be used to fine-tune such models.

Octo:An Open-Source Generalist Robot Policy



Octo is a transformer-based policy pretrained on 800k diverse robot episodes from the Open X-Embodiment dataset [Collaboration et al., 2024]. It supports flexible task and observation definitions and can be quickly finetuned to new observation and action spaces.

Octo: Architecture

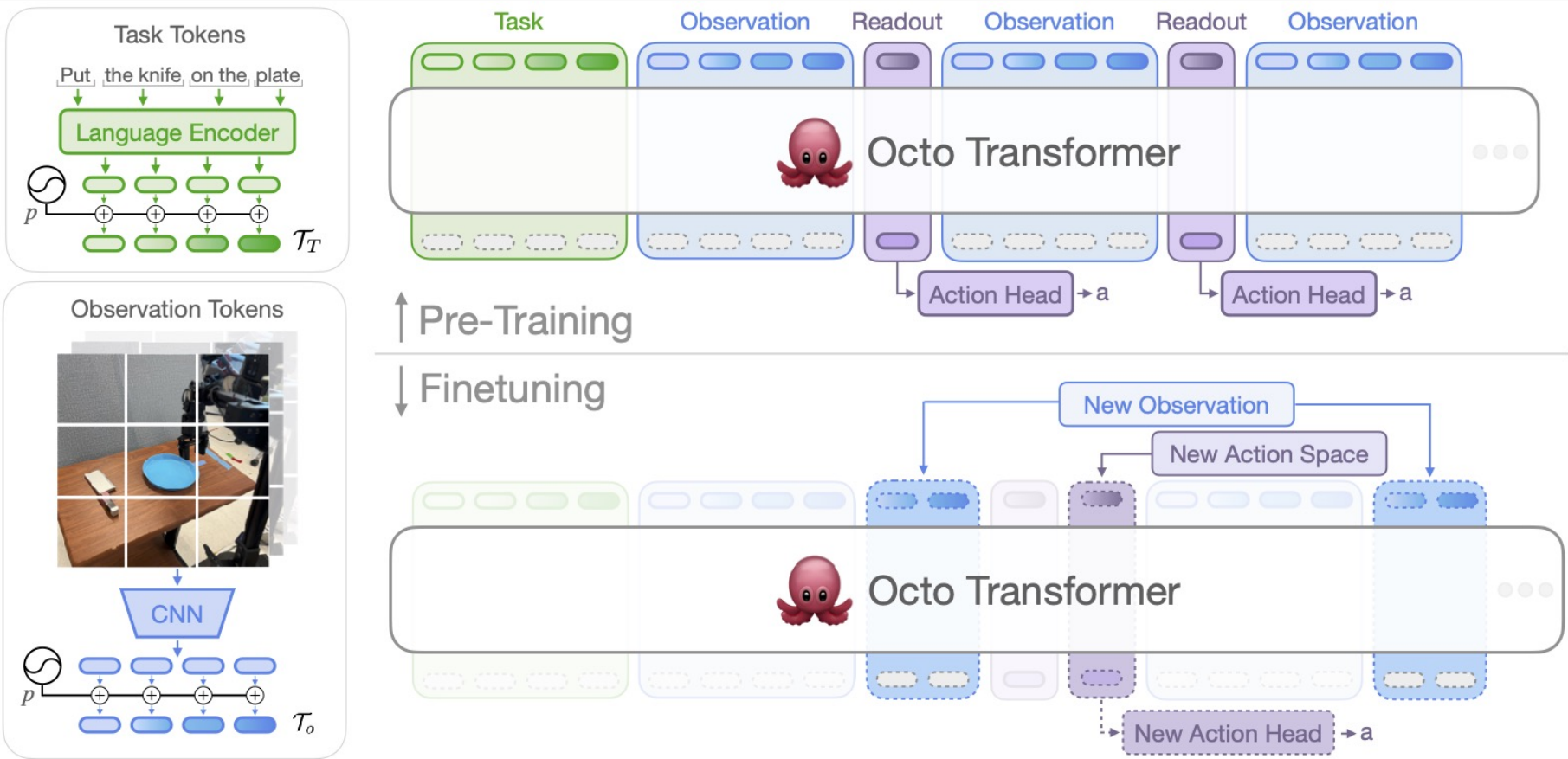


Figure from Octo Model Team et al. [2024].

Scaling the prior data: Cross-embodiment learning

Octo uses samples from a particular dataset. However, it is possible to scale the prior data using internet-scale data and vision-language models (VLM).

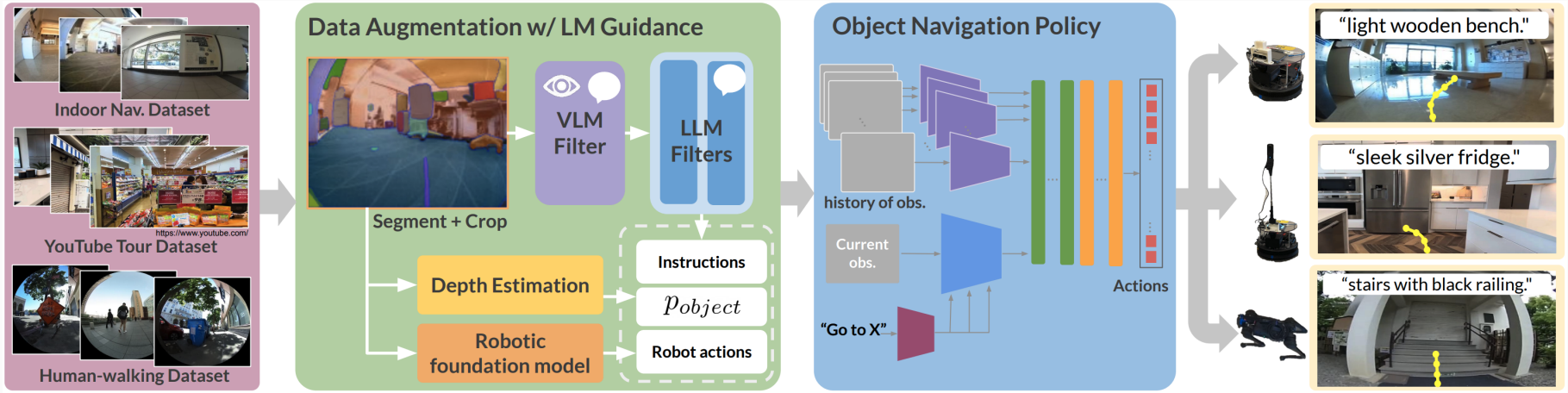
Data from different robots can be used with cross-embodiment learning.

Definition (cross-embodiment learning)

Cross-embodiment learning focuses on the ability of a single policy to perform across different robots.

Nowadays, it is usually done by determining a high-level action space common to all robots and using an abstraction layer, such as control-theory-based controllers, to simplify the control problem.

LeLaN: Foundation Model for Language-Conditioned Navigation



LeLaN leverages foundation models to label in-the-wild video data with goal language prompts and action trajectories.

Objective:

$$\min_{\theta} J(\theta) = J_{\text{pose}}(\theta) + J_{\text{col}}(\theta) + J_{\text{smooth}}(\theta)$$

where:

- J_{pose}: Encourages reaching the target object location.
- J_{col}: Discourages collisions.
- J_{smooth}: Promotes smooth velocity transitions.

$\pi 0$: generalist robot policy for dexterous manipulation

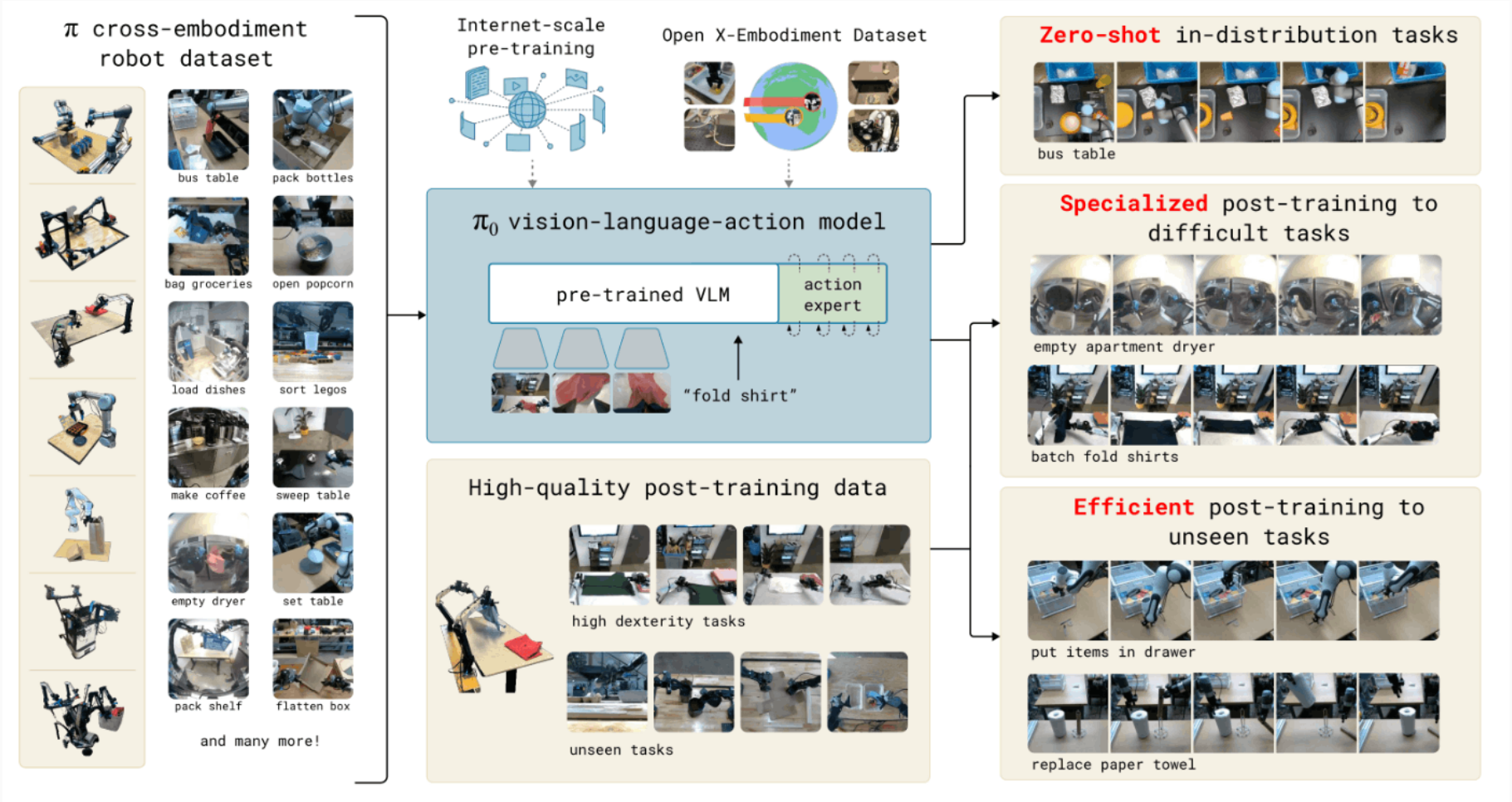


Figure from Black et al. [2024]

https://www.physicalintelligence.company/blog/pi0

Reinforcement learning with foundation models

Current foundation models such as π_0, LeLaN and Octo demonstrate that imitation-based pre-training can be good priors for autonomous control.

However, these models are limited by the dataset performance.

Future Directions:

- Efficient Fine-tuning: Combining pre-trained models with RL fine-tuning to adapt to new tasks, sensory modalities, or robot embodiments.
- Learning new tasks: Use the value function provided by VLMs to learn policies for tasks where the reward is difficult to model.

Existing gaps in reinforcement learning for autonomous control

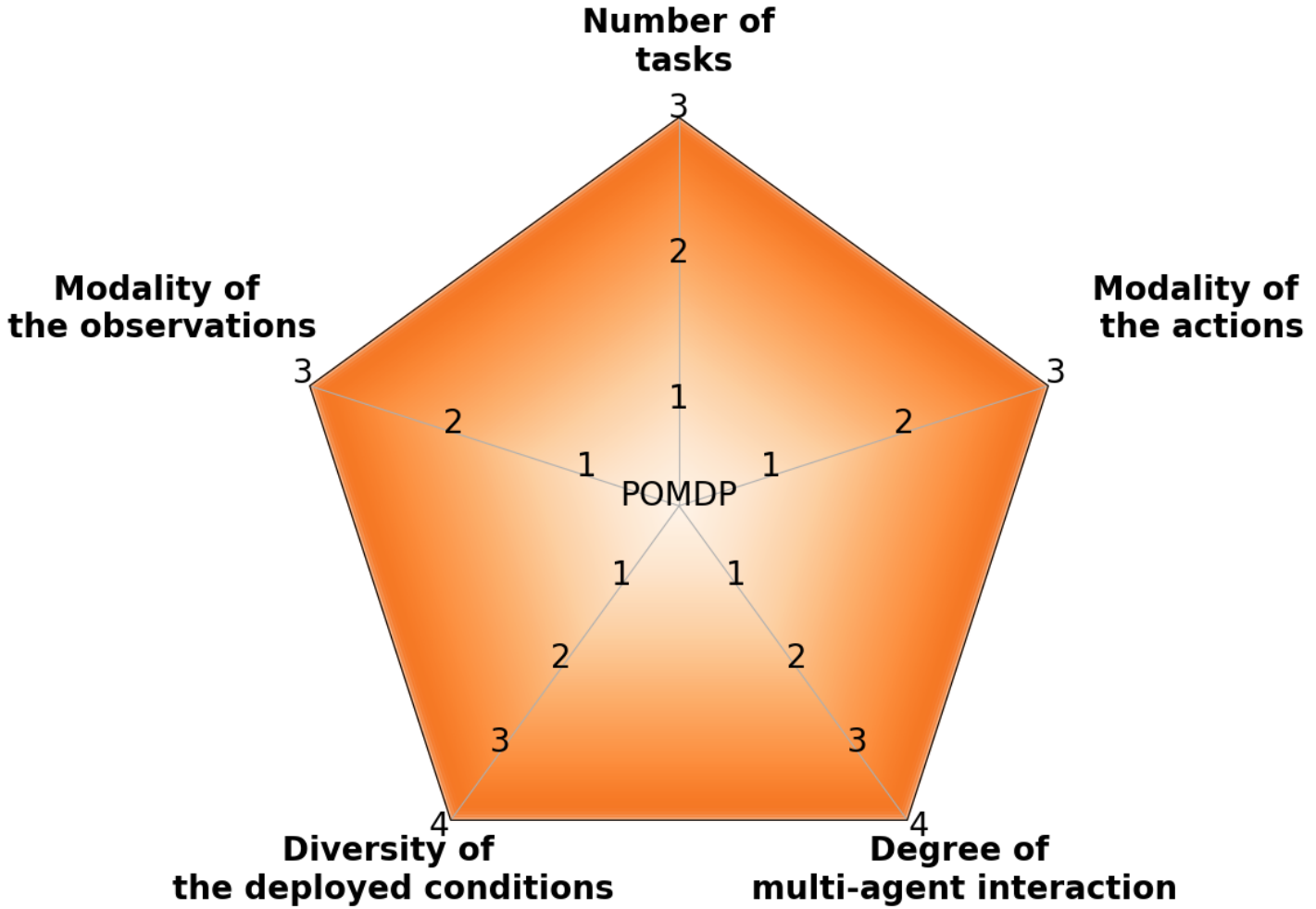
Challenges for robotic reinforcement learning (part 1)

This lecture has mainly focused on deploying good quality policies that work across the diversity of deployed conditions encountered in the real world.
We have seen how to scale these models to fuse data from multiple sensors with different frequency, how to use train models that can work with multiple effectors and how to perform well across multiple tasks.

However, we have not seen how robots can learn in multi-agent reinforcement learning scenarios: competitive, cooperative or mixed.

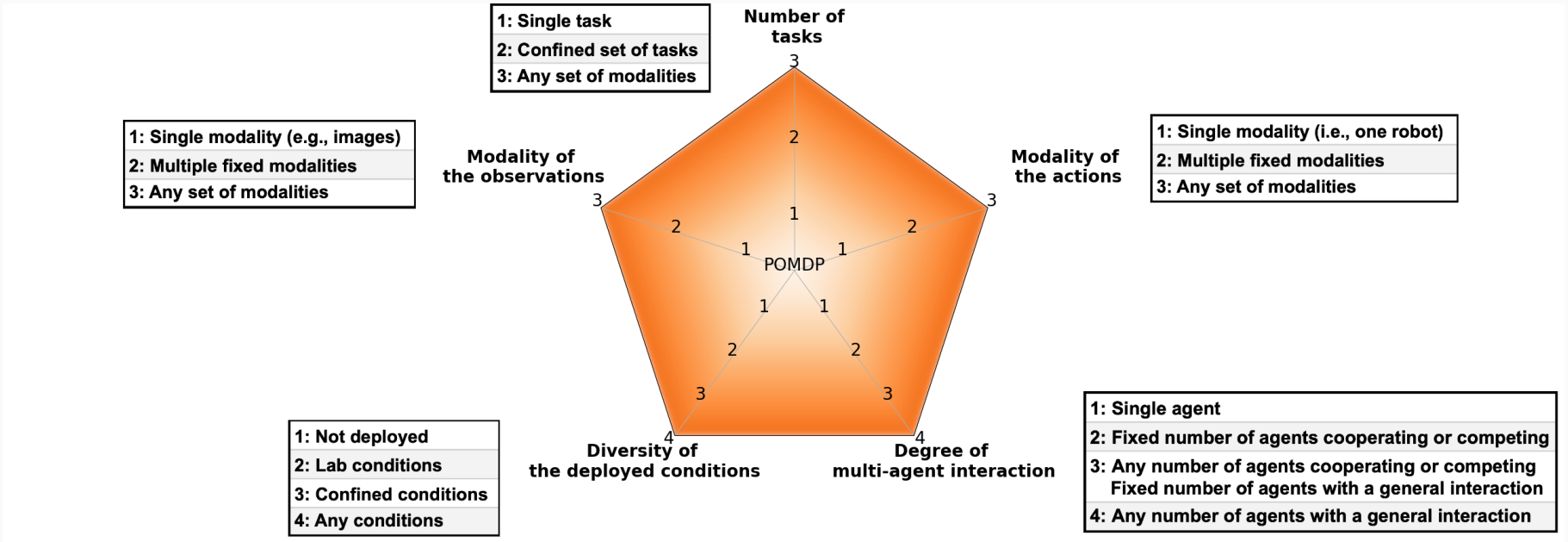
Future directions in robotic reinforcement learn should explore how to progress in these five axes of complexity together.

Challenges for robotic reinforcement learning (part 2)



Credits: Louette, A., Leroy, P., Geurts, Y., Ernst, D., (2025), Existing Gaps In Reinforcement Learning For Drone Warfare, To be submitted.

Challenges for robotic reinforcement learning (part 3)



Credits: Louette, A., Leroy, P., Geurts, Y., Ernst, D., (2025), Existing Gaps In Reinforcement Learning For Drone Warfare, To be submitted.

Final summary

Introduction

Learning in simulation

Learning in the real-world

Generalist robot policies

Existing gaps in reinforcement learning for autonomous control

References

Peter I. Corke. *Robotics, Vision & Control: Fundamental Algorithms in MATLAB*. Springer, second edition, 2017. ISBN 978-3-319-54413-7.

Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620(7976):982–987, 2023.

Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning, 2022. URL https://arxiv.org/abs/2109.11978.

Chris Gregg and Kim Hazelwood. Where is the data? why you cannot debate cpu vs. gpu performance without the answer. In *(IEEE ISPASS) IEEE International Symposium on Performance Analysis of Systems and Software*, pages 134–144. IEEE, 2011.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2019. URL https://arxiv.org/abs/1812.05905.

Simon J.D. Prince. *Understanding Deep Learning*. The MIT Press, 2023. URL http://udlbook.com.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

Takuya Hiraoka, Takahisa Imagawa, Taisei Hashimoto, Takashi Onishi, and Yoshimasa Tsuruoka. Dropout q-functions for doubly efficient reinforcement learning, 2022. URL https://arxiv.org/abs/2110.02034.

Xinyue Chen, Che Wang, Zijian Zhou, and Keith Ross. Randomized ensembled double q-learning: Learning fast without a model, 2021. URL https://arxiv.org/abs/2101.05982.

References iii

Embodiment Collaboration, Abby O'Neill, Abdul Rehman, and Abhiram Maddukuri. Open x-embodiment: Robotic learning datasets and rt-x models, 2024.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Charles Xu, Jianlan Luo, Tobias Kreiman, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. In *Proceedings of Robotics: Science and Systems*, Delft, Netherlands, 2024.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo 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_0: A vision-language-action flow model for general robot control, 2024. URL https://arxiv.org/abs/2410.24164.