Towards Scalable Robot Learning without Physical Robots

by

Younghyo Park

S.B., Seoul National University (2022)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2025

Authored by: Younghyo Park
Department of Electrical Engineering and Computer Science
May 16, 2025

Certified by: Pulkit Agrawal
Professor of Electrical Engineering and Computer Science, Thesis Supervisor

Accepted by: Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Towards Scalable Robot Learning without Physical Robots

by

Younghyo Park

Submitted to the Department of Electrical Engineering and Computer Science
on May 16, 2025 in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

ABSTRACT

The development of generalist robots—capable of performing a wide range of tasks in diverse environments—requires large-scale datasets of robot interactions. Unlike language or vision domains, where data can be passively collected at scale, robotic data collection remains costly, labor-intensive, and constrained by physical hardware. This thesis explores two complementary directions to overcome this challenge. First, we examine the limitations of training robots from scratch using reinforcement learning (RL). While RL has achieved promising results in simulation, its scalability is hindered by a largely overlooked bottleneck: environment shaping. Designing suitable rewards, action and observation spaces, and task dynamics typically requires extensive human intervention. We formalize environment shaping as a critical optimization problem and introduce tools and benchmarks to study and eventually automate this process, a necessary step toward general-purpose RL. Second, we introduce an alternative paradigm for robot data collection that does not rely on real-world robots. Using the Apple Vision Pro, we develop DART, an augmented reality (AR) teleoperation platform that streams human hand motions to cloud-hosted robot simulations. This setup enables scalable, low-latency collection of high-quality robot demonstrations without the overhead of physical setup or maintenance. Our user studies show that DART more than doubles data collection throughput while reducing operator fatigue, and policies trained in simulation using this data successfully transfer to the real world. Together, these contributions address two key bottlenecks in robot learning: the human effort required for RL environment design, and the dependence on physical robots for data. They lay the groundwork for scalable, accessible approaches to training generalist robot models.

Thesis supervisor: Pulkit Agrawal
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

It was a great two years at my time here at MIT. First, I'm grateful to my advisor, Pulkit Agrawal, for shaping my viewpoints on robotics, machine learning, research, and attitude towards life itself. I also want to thank my mentees Jagdeep Bhatia, collaborators Gabe Margolis, Lars Ankile, Antonia Bronars, and Branden Romero for the many fruitful conversations and debates we had that shaped the project and my thoughts about robotics. Lastly, I would like to thank my parents and family for their continued emotional support throughout this lonely journey.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

For decades, the idea of creating a generalist robot – one capable of human-level performance across diverse physical tasks – has lived in the realm of science fiction. But today, with the remarkable progress in language and vision models demonstrating human-like proficiency in solving diverse problems, this once-distant dream of roboticists is beginning to feel within reach.

The key to realizing this vision lies in building large-scale datasets, akin to those that catalyzed breakthroughs in language and vision. For robots, however, this presents a unique challenge. Collecting robot datasets with real robots is costly, labor-intensive, and, most critically, requires robot hardware. Unfortunately, robots are not widely deployed enough in our daily lives yet. Data collection methods that strictly require robot hardware thus create a classic chicken-and-egg problem: to make robots smarter, we need a fleet of robot hardware collecting diverse datasets at scale, but deploying such a fleet worldwide requires robots to already be smart enough to appeal as consumer products!

This chicken-and-egg problem has historically pushed most large-scale robot data collection efforts toward centralized approaches: a few research institutions investing substantial resources to purchase and setup artificial fleets of robots, setting up controlled environments, hiring dedicated data collectors and collecting datasets.

While the outcome of such efforts—decently sized datasets like [1–3]—has benefited the community, these approaches face fundamental scalability limitations. The reliance on physical robots as a prerequisite for data collection creates bottlenecks in terms of hardware costs, maintenance requirements, and geographical constraints, making it impractical to achieve the scale needed for training truly generalist robots.

The community thus started to explore alternative paths: robot training methods that do not require physical robots, and democratized, non-invasive, crowd-sourceable interfaces for robot learning. This thesis explores two such directions.

The first direction addresses the unspoken bottlenecks in reinforcement learning from scratch. While sim-to-real reinforcement learning (RL) offers a promising path to scalable robot training, its success critically depends on environment shaping: the manual design of observations, actions, rewards, and dynamics that make learning tractable. In practice, researchers spend far more effort shaping the training environment than tuning RL algorithms. My first paper argues that automatic environment shaping is an underexplored frontier and presents a formal framework and benchmarks to study this challenge. Without automating

this step, scaling RL to diverse real-world tasks remains elusive.

The second direction investigates data collection without physical robots. My recent work presents tools and platforms that leverage mixed-reality devices—such as the Apple Vision Pro—to intuitively teleoperate robots in simulation. By rendering simulated robots and scenes directly in the user's physical space using AR, and streaming hand tracking via lightweight protocols, we show that data can be collected faster, with less fatigue, and at significantly lower cost. My final paper introduces DART, a platform enabling cloud-based AR teleoperation and crowdsourced robot demonstrations, achieving over 2× higher data throughput compared to traditional teleoperation setups. All demonstrations are automatically stored in a shared public database, paving the way toward internet-scale robot data collection.

Together, these efforts point toward a more scalable future for robot learning: one that sidesteps the limitations of physical hardware, reduces the human burden in environment design, and opens the door to broad participation in data collection. By combining innovations in RL infrastructure and AR-based human interfaces, this thesis aims to reimagine how we train generalist robots.

Chapter 2

Background

2.1 Large-Scale Robot Data Collection Efforts

Addressing the need for large-scale datasets in robotics, there have been two primary approaches within the community. The first approach, as exemplified by projects like [4], focuses on gathering existing datasets from various robotics institutes worldwide into a single place. These initiatives involve a central team overseeing the data gathering, post-processing, and release. The second approach involves teams actively collecting large-scale datasets themselves by teleoperating robots in real-world environments. For example, [5] collected 110k trajectories for diverse tasks through real-world teleoperation with the help of volunteer participants. Similarly, [6] created a dataset of 60k trajectories using a low-cost robotic arm. Most recently, [3] have released 76k demonstrations across 564 scenes using a Franka Panda attached to a mobile platform. These efforts all unanimously highlight the value of large datasets in improving the performance of trained policies.

However, we argue that relying on disconnected, project-level efforts to create such datasets is not a scalable solution for the robotics community. The episodic, labor-intensive collection efforts seen in these examples fail to mirror the organic growth of language and vision datasets on the internet. Furthermore, these datasets are limited in scope, primarily focused on single-arm robots with parallel jaw grippers, neglecting the richness of bimanual or dexterous manipulations. Finally, these datasets are collected exclusively in real-world settings, overlooking the significant potential of simulation as a data source. Simulation allows for the refinement and augmentation of human-collected – and therefore possibly suboptimal – datasets through online reinforcement learning using massively parallelizable simulation environments [7]. Such refinement can address the potential performance saturation often observed on policies trained only with supervised learning [8–11].

2.2 Collecting Robotic Dataset in Simulation

Using simulation as an alternative environment for collecting demonstrations has been explored in the community. For example, [12] utilized webcams attached to laptops to allow users to teleoperate various robot morphologies in simulation. [13] employed a VR interface where humans control simulated dexterous hands, while specialized exoskeletons capture

their hand movements. More recently, with advancements in VR devices, [14, 15] have demonstrated similar technical stacks that no longer require external hand trackers, but instead utilize the built-in capabilities of modern VR/AR devices to capture hand movements. All aforementioned systems use stereo rendering streams as a source of visual feedback. However, relying on raw visual streams of simulated renderings inevitably creates a noticeable latency in network communication, forcing designers to trade-off visual fidelity and latency to maintain real-time performance. The use of Augmented Reality (AR) objects instead as a visual scene representation, on the other hand, has not yet been thoroughly explored as a solution to this problem. Finally, no existing platform has fully leveraged simulation's potential by making data collection widely accessible and available to the general public – particularly to those without specialized knowledge in robotics or the ability to set up simulation servers.

2.2.1 Augmented Reality for Robot Data Collection

Augmented Reality (AR) has been explored as a valuable tool to support the data collection process for robots. For instance, [16] leveraged mobile device AR capabilities to develop a waypoint-based teaching pendant using a virtual robot. Similarly, [17] used AR renderings to provide visual cues of robot behaviors while recording human motions in the real world. [18] also employed AR-rendered robots to guide the teleoperation process for real-world robots. However, none of these works fully leveraged AR's potential to teleoperate virtual robots in simulation through a tightly integrated control-sensory feedback loop, particularly with an emphasis on large-scale, crowd-sourced data collection.

Chapter 3

Environment Shaping Problem for Reinforcement Learning

3.1 Introduction

The advent of foundation models for speech, vision, and language processing has revolutionized Artificial Intelligence. It is widely believed that robotics is the next frontier, and the race to develop a foundation model for robotics is ongoing. The critical challenge in this pursuit is the limited availability of robotic data: trajectories of observations and robot actions. This starkly contrasts with computer vision and natural language processing, where large amounts of readily available internet data can be used. One way to gather robotic data is to deploy robots for many tasks in diverse environments. However, such deployment is only feasible if robots create value, i.e., successfully solve the task often enough. The result is a chicken-and-the-egg situation – robots need to be useful to be deployed, but for them to be useful requires collecting enough data to train a controller that creates value. The real question, therefore, is bootstrapping data collection.

A natural way to collect data is by teleoperating a robot to perform diverse tasks. However, this paradigm is challenging to scale as the human effort grows linearly with the amount of data that needs to be gathered. To ease data collection, recent works have made teleoperation more capable and easier [19, 20], but it doesn't change the linear scaling. At some point in the future, it is plausible that we will have enough data to train a large model that will reduce the number of demonstrations required to learn a new task and make the human effort sub-linear. However, we are far from that point. The primary reason is that policies obtained via supervised learning on a small dataset of demonstrations (i.e., learning from demonstration) have limited robustness and generalization and, therefore, cannot be used to collect data autonomously in more diverse settings.

In theory, given a reward function or by inferring a reward function from the demonstrations, autonomous data collection is possible using reinforcement learning (RL). It has been hard to realize this promise of RL because training in the real world often requires babysitting the robot to ensure safe operation, ensuring the reward function is not hacked, performing resets, and the data inefficiency of RL algorithms means they run for a long time before useful behaviors are discovered. Real-world RL training is an active area of research [21], and

recent work has shown the plausibility of learning locomotion behaviors from real-world RL training [22, 23]. However, real-world RL is yet to achieve state-of-the-art robotic controllers, which means the data it generates is sub-optimal.

A related line of work bypasses the difficulty of training in the real world by training with RL in simulation and then successfully deploying policies in reality (i.e., sim-to-real RL). Such training has achieved state-of-the-art behaviors across many robot morphologies and complex tasks involving legged locomotion [24–29], dexterous manipulation [30–33], drone racing [34, 35] and others. A single general-purpose RL algorithm, PPO [36], has powered these successes. Consequently, one might believe that a recipe for scaling to diverse tasks and collecting a sizeable high-quality dataset of trajectories exists. However, the reality is that immense manual task-specific modeling and engineering are required – something we call *environment shaping* (a generalization of the term reward shaping to include other environment choices to ease optimization) – to make things work, which is the primary bottleneck in our opinion. Although the environment shaping bottleneck is highly relevant to robotics, it exists in any domain where reinforcement learning is applied to real-world problems, including transportation systems [37], autonomous driving [38], finance [39], and power management [40].

We expand the previous usage of the term environment shaping [41] to include all choices such as designing reward function, curriculum, observation/action spaces, initial state distribution, reset functions performed manually to make training possible. These issues have been individually studied for a long time [42–45], but a holistic and critical analysis of the human effort required to make these choices even with the latest algorithmic advances has not been made.

Using examples from recent applications of RL to robotics, **this position paper argues that the primary bottleneck for scaling up reinforcement learning is its need for manual environment shaping.** Specifically, this is a call to action for the RL research community:

- Distinguish *modeling* from *shaping* design choices in RL environments. Many works describe the final shaped environment but not a repeatable procedure that can transfer

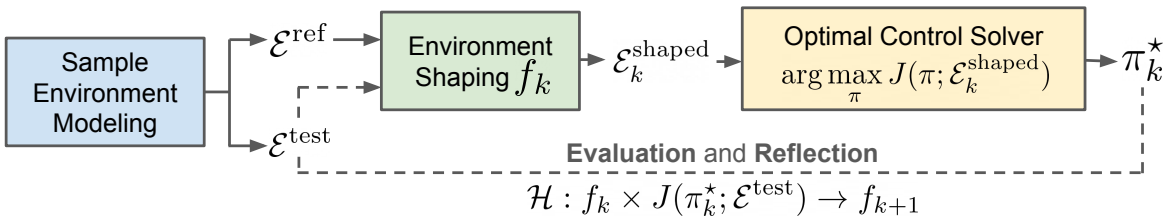


Figure 3.1: Flowchart of a typical behavior generation pipeline using reinforcement learning with simulation, illustrating four distinct subtasks of sample environment modeling, environment shaping, RL training, and outer feedback loop with behavior evaluation and reflection. We highlight the manual, task-driven **environment shaping** as a key, yet often overlooked, bottleneck in generalizing the success of RL. We thus advocate for automating the environment shaping process to broaden RL's applicability.

the same shaping to a new robot or task.

- Prioritize research into *automatic environment shaping* as a pathway to generalize domain-specific successes in RL.

- Prioritize *benchmarks* that measure the total expense of applying reinforcement learning to real-world robotics tasks, by including environments with 'unshaped' versions and corresponding human-shaped baselines. The point being that existing benchmarks have already performed environment shaping (i.e., hide the true problem) and, therefore do not serve as good candidates for further research into better algorithms. [1]

3.2 Robotic Behavior Generation with RL

To support a precise definition of environment shaping, we first describe a typical workflow for generating robotic behaviors using reinforcement learning (RL) in simulation (Figure 3.1).

We decompose behavior generation into four subtasks; sample environment generation (Sec 3.2.1), environment shaping (Sec 3.2.2), RL training (Sec 3.2.3), and the outer feedback loop with behavior evaluation and reflection (Sec 3.2.4). In delineating these substages, we pinpoint typical human efforts involved in the process.

3.2.1 Modeling Sample Environments

Consider $\hat{p}(e)$ as the oracle distribution of the target environment we want to deploy our robots in. Our goal is to generate behavior that is performant (with respect to objective J) and robust under $\hat{p}(e)$,

$$\max_{\pi} E_{\hat{e}\sim\hat{p}(e)} J(\pi; \hat{e})$$

The target environment can either be a specific real-world environment that already exists (e.g. kitchen at a specific location) or a generic concept (e.g. typical household kitchen).

Unfortunately, it's extremely difficult to model this oracle distribution either way; it requires comprehensive knowledge of all possible environmental variables and conditions, which is often infeasible due to the complexity, variability, and limited observability in the real world. Innate vagueness of generic concepts is often an issue as well. Just imagine modeling a true oracle distribution of a chaotic real-world dishwasher in simulation! (Figure 3.2)

In contrast, modeling a *single* sample environment, $\hat{e}$, a specific instance drawn from oracle environment distribution,

$$\hat{e} \sim \hat{p}(e),$$

and importing that to a simulation is much more feasible. This is why robotics practitioners typically start the behavior generation process by first designing a single sample environment: (a) modeling and importing robots and necessary assets in simulation, and (b) manually placing them in their default poses. We often generate a *set* of those sample environments to kick things off. This is what practitioners do for the first blue box in Figure 3.1.

[1] https://auto-env-shaping.github.io/

Figure 3.2: Example of environment complexity: an overloaded and disorganized real-world dishwasher.

Such a set of sample environments actually serves a purpose: it is a useful representative testbed environment that can be used to estimate the behavior performance under the true oracle distribution $\hat{p}(e)$. Combined with any form of task specification r [46], we define a simulated testbed $\mathcal{E}^{\text{test}}$ where the trained behaviors can be evaluated.

Definition 3.2.1 (Test Environment) *Let $\{\hat{e}_1, \cdots, \hat{e}_n\}$ be a set of n sample environments each independently drawn from oracle environment distribution $\hat{p}(e)$. The simulated counterparts are denoted as $\hat{e}_{sim,i}$. Given a task specification r, a test environment $\mathcal{E}^{test}$ is defined as a set of tuples:*

$$\mathcal{E}^{test} = \{\langle \hat{e}_{sim,i}, r \rangle\}_{i=1,\cdots,n},$$

where the generated behavior π will be evaluated in.

Meanwhile, to prevent the behavior from overfitting to a few sample environments within $\mathcal{E}^{\text{test}}$, one can maintain two sets of sample environments; the test environment can be strictly held out from the rest of the behavior generation pipeline, letting it serve the sole purpose of behavior evaluation. A distinct set of sample environments then can be used to effectively guide the remaining design choices. We call that a *reference* environment, $\mathcal{E}^{\text{ref}}$, as illustrated in Figure 3.1.

Definition 3.2.2 (Reference Environment) *A Reference Environment $\mathcal{E}^{ref}$ is a distinct set of sample environments that provides useful context to the shaping algorithm. Trained behaviors will not be evaluated here to avoid overfitting.*

For instance, when designing a robotic behavior for unloading a dishwasher, sample environments would include multiple instances of dishwashers loaded with varying configurations of dishes and utensils. Sampled configurations then could be split into a set of reference environments for guiding the shaping and a set of held-out test environments for evaluation. This diversity in configurations provides the shaping algorithm with a broad context, incentivizing it to infer the underlying distribution of object placements. The goal is

for the reinforcement learning (RL) trainer to sample from this inferred distribution during training, ensuring that the generated behaviors are robust and adaptable to various real-world scenarios without overfitting to a specific set of environments.

3.2.2 Shaping Reference Environments

A straightforward subsequent step of behavior generation might be to directly use the reference environments $\mathcal{E}^{\text{ref}}$ as an RL environment, with the expectation that algorithms like Proximal Policy Optimization (PPO) [36] will find performant and generalizable behaviors. However, this approach often falls short due to the inherent sparsity of these environments.

Reference environments often present a challenging optimization landscape for RL algorithms due to their *sparse* nature. For example, it might be rare to obtain nonzero rewards or the observation space might be dominated by spurious features that encourage poor local minima. To mitigate these challenges, human engineers typically go through a process of *shaping* the elements of $\mathcal{E}^{\text{ref}}$. This modification, aimed at enhancing the *learnability* of the environment, includes introducing denser learning signals and additional modifications encouraging effective exploration. The resulting *shaped environment* [41] then used as a training ground for the subsequent optimal control solver.

Definition 3.2.3 (Shaped Environment) *A Shaped Environment $\mathcal{E}^{shaped}$ is a modification of a reference environment, i.e., $\mathcal{E}^{shaped} = f(\mathcal{E}^{ref})$. The transformation f incorporates design choices specifically optimized for learning performance, smoothing the optimization landscape enabling the solver to find better solutions with maximal performance in $\mathcal{E}^{test}$. The transformation function*

$$f : \mathcal{E}^{ref} \to \mathcal{E}^{shaped} \tag{3.1}$$

is defined as shaping function.

The *shaping* process for a robotics environment usually involves manually explored design choices, including shaping the reward [43, 44], modifying the action space [47, 48], designing curricula on the environment dynamics, initial state distributions, and goal distributions [42, 49–51], crafting the state space [52], and shaping the right failure conditions for early termination [41]. We describe details of common shaping operations in Sec 3.3.1.

3.2.3 RL Training

Once the shaped environment is obtained, the next step of behavior generation is to use RL algorithms, i.e., PPO [36], to find a behavior π that best performs on the shaped environment $\mathcal{E}^{\text{shaped}}$. Formally, the algorithm aims to find an optimal behavior π for the following optimization problem:

$$\begin{aligned} &\max_{\pi} E_{\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t r_t(s_t,a_t))\right] \\ &\text{s.t.} \quad s_{t+1} \sim p(s_t,a_t;\mathcal{E}^{\text{shaped}}). \end{aligned} \tag{3.2}$$

While the RL training process also requires a range of design decisions, such as algorithmic choices and hyperparameter adjustments, these areas are relatively well-researched and documented [53, 54].

However, we note that in a pratical context of robotic behavior generation, tuning the RL setting (e.g. neural architecture search for policy or hyperparameter tuning) is often underprioritized compared to the effort put into environment shaping. In IsaacGymEnvs [7], for instance, simple Multilayer Perceptron (MLP) networks are employed, and off-the-shelf RL algorithm implementations are utilized with their default configurations. This shows that algorithms like PPO and their default settings are capable enough when paired with *ideally shaped* environment.

3.2.4 Optimizing Environment Shaping via Iterative Behavior Evaluation and Reflection

Once an optimal behavior $\pi^\star$ is obtained via RL training, the behavior is *evaluated* on the test environment $\mathcal{E}^{\text{test}}$ and *reflected* by human engineers. Denoting the reflection process as $\mathcal{H}$ of analyzing the generated behavior $\pi^\star$ in test environment $\mathcal{E}^{\text{test}}$ and coming up with a better environment shaping f,

$$\mathcal{H} : f_k \times J(\pi_k^\star; \mathcal{E}^{\text{test}}) \to f_{k+1}, \tag{3.3}$$

robotic behavior generation process can be formally defined as an **iterative optimization process** over the **environment shaping function** f,

$$\begin{aligned} f_{k+1} &= \mathcal{H}\left(f_k, J(\pi_k^\star; \mathcal{E}^{\text{test}})\right) \\ \text{where} \quad \pi_k^\star &= \underset{\pi}{\text{argmax}}\ J(\pi; \mathcal{E}_k^{\text{shaped}}), \\ \mathcal{E}_k^{\text{shaped}} &= f_k(\mathcal{E}^{\text{ref}}), \quad f_0 = \mathbf{I}_{\text{identity}} \end{aligned} \tag{3.4}$$

which aims to find an optimal shaping function $f \in \mathcal{F}$ for the following bi-level optimization problem:

$$\begin{aligned} &\max_{f \in \mathcal{F}}\ J(\pi^\star; \mathcal{E}^{\text{test}}) \\ &\text{s.t.} \quad \pi^\star \in \arg\max_{\pi} J(\pi; \mathcal{E}^{\text{shaped}}), \quad \mathcal{E}^{\text{shaped}} = f(\mathcal{E}^{\text{ref}}). \end{aligned} \tag{3.5}$$

After shaping is applied, the new environment may not reflect the original task; therefore, note that the outer level of the bilevel optimization maximizes $J(\pi^*; \mathcal{E}^{\text{test}})$ which is the return evaluated in the original test environments without any shaping. If the inner level optimizes a shaped environment well, but with poor correspondence to the original task, it will be dispreferred by the outer loop.

3.3 The Current State of Environment Shaping

Having established the role of environment shaping operations in successful behavior generation, we now probe deeper into the unique challenges that the problem of environment shaping and its optimization procedure presents. Specifically, we make the following arguments with supporting experiments and analysis:

```python
def shaped_action_space(self):
  # scale the targets by the joint limits
  self.cur_targets = scale(self.actions,
         self.shadow_hand_dof_lower_limits,
         self.shadow_hand_dof_upper_limits)

  # compute the moving average of targets
  self.cur_targets = self.alpha * self.cur_targets +
                (1 - self.alpha) * self.prev_targets
  self.cur_targets = tensor_clamp(self.cur_targets,
              self.shadow_hand_dof_lower_limits,
              self.shadow_hand_dof_upper_limits)
  self.prev_targets = self.cur_targets[:]

  # compute the torques according to PD control law
  torque = 3 * (self.cur_targets - self.shadow_hand_dof_pos) - 0.1 * self.
   shadow_hand_dof_vel

  # apply torques
  self.gym.set_dof_actuation_force_tensor(self.sim, gymtorch.unwrap_tensor(torque))
```

```python
def unshaped_action_space(self):
  # directly apply the prediction action as torques
  self.gym.set_dof_actuation_force_tensor(self.sim, gymtorch.unwrap_tensor(self.actions))
```

Figure 3.3: Action space shaping: (Top) Original shaped action space with task-specific features. (Bottom) Unshaped action space consisting of joint torque commands. Some shaped code has been slightly modified from the source to increase brevity and clarity while preserving the original logic.

- Popular RL benchmark environments are *artificially* made easy for RL with task-specific environment shaping. We should benchmark our algorithms in unshaped environments if we want them to solve new problems without task-specific environment shaping step. (Section 3.3.1)
- Shaping multiple attributes of the environment at once (reward, observation space, action space, etc.) is a tricky, non-convex optimization problem (Section 3.3.2).
- Reward shaping is not the only problem. Existing automation efforts focus too narrowly on rewards (Section 3.3.3).

3.3.1 RL Benchmarks for Robotics are Artificially Easy

Benchmark environments for robot reinforcement learning include task-specific environment shaping to make it feasible to help baseline RL algorithms perform reasonably well. However, these modified environments might not fully assess how RL algorithms progress towards solving various control problems independently, treated as a black-box, without needing task-specific adjustments.

In this section, we outline the common task-specific design choices associated with different aspects of environment shaping, while formally defining what an *unshaped* counterpart — one with minimal or no human engineering required — would look like. We consider a case study of the IsaacGymEnvs task suite [7].

AllegroHand	Reward	Change
all shaped	38777	–
sparse reward	0	↓ 38777
unshaped action space	21530	↓ 17247
unshaped observation space	2114	↓ 36663
no early termination	0	↓ 38777
single initial state	0	↓ 38777
single goal state	141155	↑ 102378

Humanoid	Reward	Change
all shaped	7554	–
sparse reward	5237	↓ 2317
unshaped action space	67	↓ 7487
unshaped observation space	0	↓ 7554
no early termination	705	↓ 6849
single initial state	5735	↓ 1819

Anymal	Reward	Change
all shaped	−45	–
sparse reward	−2789	↓ 2744
unshaped action space	−2499	↓ 2454
unshaped observation space	−2656	↓ 2611
no early termination	−43	↑ 2
single initial state	−17	↑ 28
single goal state	−2516	↓ 2470

Table 3.1: **Impact of environment shaping on policy optimization.** Removing task-specific design choices in the reward, action space, state space, early termination, or initialization incurs performance reductions. Top row: environment with original *shaped* design choices. Each subsequent row shows performance after training with a corresponding unshaped design choice. The performance of all policies is evaluated in a fully unshaped environment.

Action Space.

How would an *unshaped* action space look, and how *shaped* is the action space in the case environments? For a typical robot in rigid multibody simulation, the unshaped action space would be the motor torques: passing in the policy output a directly to the motor as a torque τ — no scaling or transforming the outputs, no gains to be tuned.

Designing a shaped action space for a robot is thus equivalent to the problem of choosing a low-level controller (and its corresponding parameters) that converts a policy output $\mathbf{a}$ with different physical meanings into an executable motor torque τ that can drive the actuators. The low-level controller can implement a prior like whether to resist or comply to external forces. Examples include proportional-derivative control, differential inverse kinematics controller, operational space control [55], or impedance control [56]. These are projections of the policy outputs onto a strict subset of the space of possible torque commands. The

```python
def shaped_observation_space(self):
  root_states = rigid_bodies[:, self.root_handle]
  base_quat = root_states[:, 3:7]
  # base linear velocity (from local frame)
  base_lin_vel = quat_rotate_inverse(base_quat, root_states[:, 7:10]) * lin_vel_scale
  # base angular velocity (from local frame)
  base_ang_vel = quat_rotate_inverse(base_quat, root_states[:, 10:13]) * ang_vel_scale
  # transformed base orientation
  projected_grav = quat_rotate(base_quat, gravity_vec)

  # scaling/normalizing values
  commands_scaled = scale(commands,
                [lin_vel_scale, ang_vel_scale])
  dof_pos_scaled = dof_pos_scale *
                (dof_pos - default_dof_pos)
  dof_vel_scaled = dof_vel_scale * dof_vel

  # concatenate the relevant features
  obs = torch.cat([base_lin_vel, base_ang_vel,
                   projected_grav, commands_scaled,
                   dof_pos_scaled, dof_vel_scaled,
                   actions], dim=-1)
```

```python
def unshaped_observation_space(self):
   # include entire unprocessed simulator state
   obs = torch.cat([dof_pos,
               dof_vel, torques,
               rigid_bodies.reshape(num_envs, -1),
               rigid_body_force.reshape(num_envs, -1),
               commands, actions], dim = -1)
```

Figure 3.4: State space shaping: (Top) Original shaped state space with task-specific features. (Bottom) Unshaped state space contains the entire raw simulator state.

choice of action space can significantly influence the performance of learning algorithms in the environment [47].

In IsaacGymEnvs, there are multiple distinct cases of shaped action spaces. `AllegroHand` uses joint position targets with moving average smoothing as its action space (See Figure 3.3 for example code). For `Anymal`, the relative joint position target is used as an action space with proportional-derivative controller. For `Humanoid`, scaled joint torques are used. Table 3.1 empirically shows that removing these *shaping* operations on action space largely impacts the training performance.

Observation Space.

What is an *unshaped* observation space, and how did IsaacGymEnvs *shape* the observation? Crafting a *shaped* observation space involves strategic selection and transformation of variables from an unshaped observation — the entire state in simulation that's been exposed to the user. The process, integral to optimizing policy learning performance, includes (a) useful transformation of the unshaped variables (i.e., features) and (b) discarding unnecessary variables. Figure 3.4 shows the details of the shaped state space in the IsaacGymEnvs `Anymal` environment.

Different choices of observation space shaping can substantially impact robot performance. In the setting of quadrupedal locomotion, [52] found that different aspects of the task (robustness, performance) are sensitive to different features included in the observation space.

Table 3.1 shows our findings agreeing that observation shaping has a significant impact on training performance. For `Humanoid`, for instance, using an unshaped observation space makes the task completely unlearnable.

Reward.

The *unshaped* reward represents the true objective function we actually wish to optimize, often defined as extrinsic reward [44, 57] or task-fitness function [58].

Shaping the *extrinsic* reward into a more informative *intrinsic* reward to make it more conducive to RL algorithms is a well-studied topic in the community [43, 57, 59, 60]. The shaping procedure is usally designed to provide *denser* learning signals and to prevent the policy from overly exploiting the innate vagueness of extrinsic reward, i.e., reward hacking. It typically involves a few commonly used strategies. One popular strategy is to reward *progress* towards the goal by adding distance metrics measuring how close the agent is to reaching it. If some attributes of successful behavior are known beforehand, those terms can be added to the shaped reward [61, 62]. If a trajectory of a successful behavior is known, one can also compute a *similarity* between the generated behavior and the trajectory and try to maximize the similarity [63]. Offering bonuses for exploring new states [64] is also known to facilitate RL training. Table 3.1 shows that *shaping* reward has significant impact on the training performance.

It's worth noting, however, that defining the extrinsic reward itself poses its own set of challenges, which is well addressed in [46]. The task fundamentally requires translating human intentions and goals into numerical values that an algorithm can optimize, often posing significant challenges. In the scope of this position paper, we presume the task has been defined and consider the challenge of learning a policy to perform it.

Initial/Goal State.

Let's consider a concrete example: how would you *shape* an initial state of a dishwasher to make the behavior robust under the chaotic randomness typically exhibited in real-world (Figure 3.2)? Where would you even start shaping things from? How would an *unshaped* counterpart look like for initial/goal state shaping?

A reasonable *unshaped* intial/goal state to start with are the nominal states defined in *reference* environment – manually designed sample environment with every actors (robots and assets) staying in its nominal pose. We often assume such nominal pose to be a mean of its underlying distribution, commonly assumed as Gaussian or Uniform distribution. This is indeed the most simple yet common technique of designing initial state distribution for many robotics tasks – we just randomly perturb robot joints and assets around its nominal (reference) pose!

When the task gets more complex, however, this simple approach starts to break quickly. Imagine randomly perturbing a single nominal state of a dishwasher, or randomly perturbing the nominal pose of a quadruped standing on a rough terrain; dishes and ladles, feets and terrain will be in penetration most of the time. Doing rejection sampling can be a stopgap solution, but it might end up rejecting most of the samples, making the approach nearly unusable.

In practice, robotics engineers thus take a clever, but heavily heurstic, task-dependent approach to shape initial states. To randomly initialize a quadruped on a rough terrain, for instance, people make the robot walk off a small region of flat ground [65], letting the simulation engine figure out the phsyics constraints. To generate random initial states for cluttered bin-picking tasks, we often drop objects from height in random order. To obtain initial states to train a fall-recovery policy for humanoid, we drop the humanoid from height [63]. For in-hand manipulation tasks, to obtain a diverse set of downward-facing initial grasps to start with, we first train a grasping policy and execute it to generate diverse initial starting configurations that does not drop the object immediately [66]. The task-specific nature of such strategies poses challenges in automating this shaping operation.

Moreover, designing *how* we sample from the shaped initial/goal distribution can naturally set up a learning curriculum that progresses from simpler to more challenging tasks. Take the example of training a quadruped to run at high-speed. Training only with high-speed commands could hinder the learning process. Instead, shaping a goal distribution to be a wider distribution then the actual target speed, and designing the sampling strategy to begin with slower ones and gradually increasing, can facilitate more effective learning of fast running behaviors [67].

Terminal Condition.

A strict definition of *unshaped* terminal condition might be to never terminate, resembling how the real world never terminates and never gets a chance to reset from the beginning. However, since this poses quite challenging problem for most learning algorithms, one more common choice of *unshaped* terminal condition that is fairly environment agnostic is to set a predefined episode time limit.

Shaping the terminal condition thus corresponds to deploying strategies of *early termination*, which is known to significantly improve training performance [41]. In IsaacGymEnvs, `Anymal` terminates when non-foot bodies contact the ground, `Humanoid` terminates when the torso falls below a certain height, and `AllegroHand` terminates when the object falls off from the hand. Table 3.1 shows that early termination significantly improves the training performance in all three tasks.

3.3.2 Shaping the Entire Environment is Harder than Shaping One Component

In the previous section, we saw that policy learning can be highly sensitive to each individual aspect of environment shaping (Table 3.1) and that the shaping choices in IsaacGymEnvs vary qualitatively depending on the task. This motivates that, to promote the development of truly automatic behavior generation, we should consider benchmarking against a suite of unshaped environments. If an algorithm can learn policies in unshaped environments, it should be applicable to newly defined tasks and environments without requiring additional manual shaping on those environments. The unshaped environments are an appropriate benchmark for methods that couple automatic environment shaping with RL, or to directly attempt to solve by improving RL algorithms.

What kinds of methods might we try using to optimize an entirely unshaped environment? If we aim to accomplish this by introducing shaping, we may have to search over not just one aspect but all aspects of shaping. To understand the optimization landscape, suppose we have access to an oracle that proposes the human-designed environment shaping as a candidate, and we use this to perform a hill-climbing search in one aspect of the environment at a time: first, optimize the observation space, then the action space, then the reward function, and repeat until convergence. Figure 3.5 illustrates the consequence of this strategy in three environments from IsaacGymEnvs. Each node represents a shaped version of the environment. The node's color indicates the trained policy's performance on the shared test environment (unshaped). Edges indicate environments that differ by one type of change (state space, reward, etc.), and the bold arrows indicate the path taken by decoupled hill-climbing on one change at a time. We found that each environment has multiple local maxima where the hill climbing would get stuck in a suboptimal configuration. These local maxima differ in at least two (as many as four!) types of shaping. This suggests that the environment shaping design space is non-convex in the different types of shaping. Therefore, we should aim to develop techniques that consider shaping all parts of the environment jointly.

3.3.3 Existing Automation Focuses Narrowly on Rewards

There have been prior attempts to partially automate the process of environment shaping. However, most of the efforts have focused on the subtask of reward shaping. The concept of reward shaping was formalized by [43] and the idea of automated reward shaping through evolutionary search was advanced by [44, 57]. Recent works have formulated the problem of reward shaping as bilevel optimization with the environment design at the top level and policy learning at the bottom level and used LLMs [68–70] or gradient-based methods [71] to generate candidates.

LLM-based methods can benefit from prior knowledge about coding and robotics derived from internet training data to act as an efficient sampler for generating candidate shaped rewards expressed as code. Since other aspects of the environment shaping can also be expressed as code, it is straightforward to test how these methods extend. We evaluated the LLM-based reward shaping algorithm Eureka [69] at the task of designing the observation and action space for the `Anymal` IsaacGymEnvs environment. As in the original application to reward shaping, we generated five generations of 16 parallel candidate shaping functions using GPT-4, including the best performing candidate from each generation in the prompt for the next generation. Eureka succeeded in designing effective observation and action spaces for this environment (Table 3.2) while all other aspects of the environment were pre-shaped.

Motivated by Section 3.3.2, we also tested whether Eureka can jointly optimize multiple aspects of the environment. We found that performance drastically drops when Eureka is tasked to jointly optimize multiple environment aspects, i.e., shaping both the reward and observation, even though it could optimize each individually. This suggests that shaping multiple environment aspects jointly is yet an open problem.

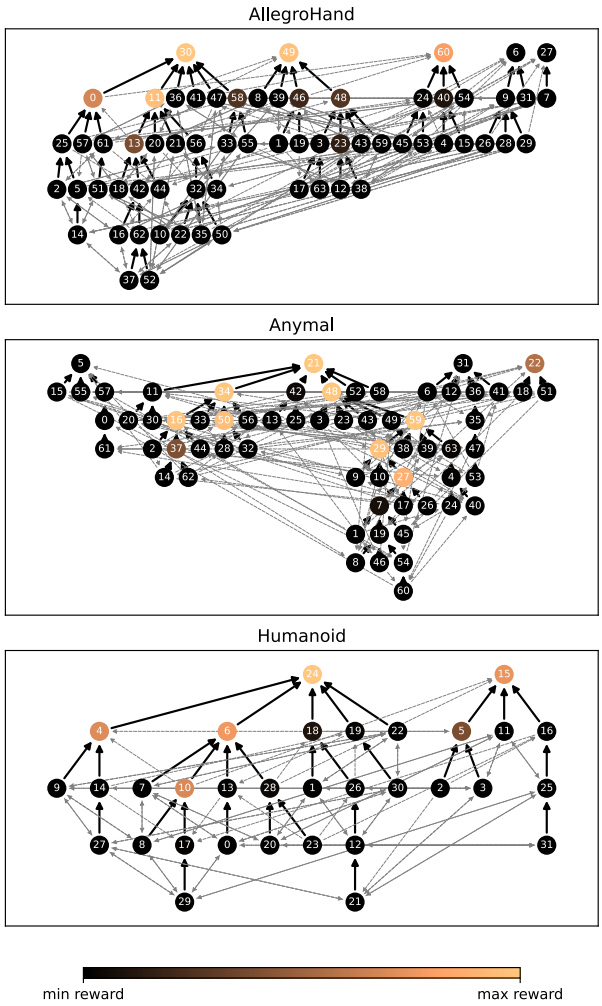


Figure 3.5: **Local optima in environment shaping problems.** Each node represents a shaped training environment. Edges connect environments that are separated by modifying one type of shaping (action space, state space, reward function, initial state, goal, or terminal condition). Bold arrows represent optimal choices for hill climbing. Each environment is shown to have multiple local optima corresponding to the top row of nodes.

3.4 Paths Forward to Automated Environment Shaping

Today, no algorithm can solve diverse unshaped tasks, although we know them to be solvable with environment shaping by human designers. How can we fix this and generalize successes in RL for robotics? There are a few possibilities:

Scale up computation.

Existing bi-level search techniques like [69] can be extended to design environment shaping and run with more compute resources to search over a greater number of candidate shaping designs. Massively parallel simulation has led to realistic robotics environments that can train quickly [7, 65]; However, some state-of-the-art sim-to-real methods take weeks to train

shaping component	Eureka [69]	Human Design [7]	Automation Performance
*reward	0.986	0.973	$\uparrow$ 0.013
†observation	0.967	0.973	$\downarrow$ 0.006
†action	0.982	0.973	$\uparrow$ 0.009
$^{\circ}$reward $\times$ observation	0.196	0.973	$\downarrow$ 0.777
$^{\circ}$reward $\times$ action	0.536	0.973	$\downarrow$ 0.437
$^{\circ}$reward $\times$ observation $\times$ action	N/A	0.973	N/A

Table 3.2: Evaluating [69] for *reward shaping, †shaping different components, and $^{\circ}$coupled shaping. `Anymal` task [7] is used as an environment. Automation performance is measured by the relative gain obtained by automating the shaping process, compared to design choices made by humans [7]. Behavior is evaluated with the following task specification: $r = \exp(\text{negative distance to command})$. 5 iterations of outer loop for Eureka. Used latest GPT-4 model. Note that GPT-4 failed to generate working code for results labeled N/A.

a single policy due to high data requirements or expensive subroutines [29, 31]. This would make performing much more outer-loop search impractical.

Improve priors.

If we can't search over more candidates, a better way is to generate higher-quality candidates more quickly. One possibility is that improved foundation models will zero-shot generate better candidate shaped environments [72, 73]. However, it's hard to predict how much this will help. Another good idea is to *mechanistically understand the strategies of human designers*. By what mechanism does the choice of observation space or curriculum improve performance? [41] proposed a holistic view of environment shaping ('Ecological Reinforcement Learning') and studied the impact of stochasticity, goal distribution, and early termination design on learnability. [52] examined the impact of observation space on learning quadruped locomotion and offered an explanation of how different observations can work better for different subtasks. [45] surveyed a number of works across applied reinforcement learning, comparing practitioners' design choices. [47] analyzed a set of common action spaces for robot control and [48] developed metrics to explain the performance gap. [74] found that constraints require less tuning than rewards to transfer across diverse robots. If more understanding is documented in these areas, improved biases could be implemented as a prior, e.g., in an LLM's context.

Shape online.

Instead of iteratively improving on the environment shaping f across training runs, can we improve it dynamically within a RL training loop? If, for example, multi-objective reinforcement learning or bilevel optimization can trade off the coefficients of multiple reward terms [57, 75, 76], then perhaps the LLM search can be performed only over the form of the different reward terms, and their relative weighting can be optimized at runtime. In other aspects of the environment, an analogous approach would be to generate a parameterized

shaping operator and automatically tune its parameters. For example, the scale of the observation or the termination height could be assigned as parameters for online optimization.

A Robotics Benchmark for Environment Shaping. Instead of evaluating RL algorithms on environments pre-shaped to work with PPO, the community should evaluate them on unshaped environments. These environments will be *too hard* to solve with existing RL algorithms. Thus, it will be necessary to (a) study how to modify aspects of the environment to make it efficiently solvable with RL and / or (b) develop better RL algorithms that can handle such challenging sparse environments.

We modified the environments in IsaacGymEnvs to use unshaped design choices to provide a good proxy for the task-agnostic output from procedural environment generators. To facilitate environment shaping research, our code exposes an API for modifying the environment code, which allows the optimizer to transform the reward, observation space, action space, etc. by editing Python functions at runtime. The API is designed so that any language models can be easily integrated to perform such transformations. Our implementation also facilitates faster evaluation of multiple environment shaping choices by training multiple policies in a single process, leveraging parallel simulation.

3.5 Conclusion

Reinforcement learning has long promised to solve decision-making problems in a task-agnostic manner. It has found great success in solving challenging but narrowly-scoped tasks in robotics. In this position paper, we argued that the key bottleneck for scalability of RL is a limited mechanistic understanding of task-specific engineering (*environment shaping*) that transforms environments to be solved more easily and is universal across domains and benchmarks. We proposed a formal definition of environment shaping as an optimization problem and identified instances of shaping in robotic tasks. Finally, we identified key steps forward such as developing computationally efficient search over environment shaping; improving our tools for implementing such shaping and understanding its impact on learning dynamics; and defining benchmarks for this problem. We hope this will motivate an increased focus in RL research on communicating and evaluating environment-shaping measures that impact performance rather than solely emphasizing the impact of the learning algorithm or policy architecture.

Chapter 4

Using AR Devices to Control Robots and Collect Data

4.1 Introduction

Apple Vision Pro (AVP), a virtual/augmented reality (VR/AR) device recently released by Apple, tracks various movements of the user wearing the device. Wrist and finger movements, for instance, are constantly tracked and used as its primary interface for user interaction; sensors included in the device are optimized to provide an accurate tracking of human movements. Such tracking capability makes the device particularly appealing for many robotic applications: AVP can be used to (1) record the navigation and manipulation behaviors of humans in real-world environment, and can also be used to (2) teleoperate a robot with intuitive human motions. The device's virtual and augmented reality capabilities also opens up new avenues for immersive experiences during robotic teleoperation.

This repository thus aims to provide diverse array of tools for using AVP in robotics applications, starting with an easy-to-use library with minimal dependencies that can stream the tracking data from the Vision Pro to any client device connected to the same network.

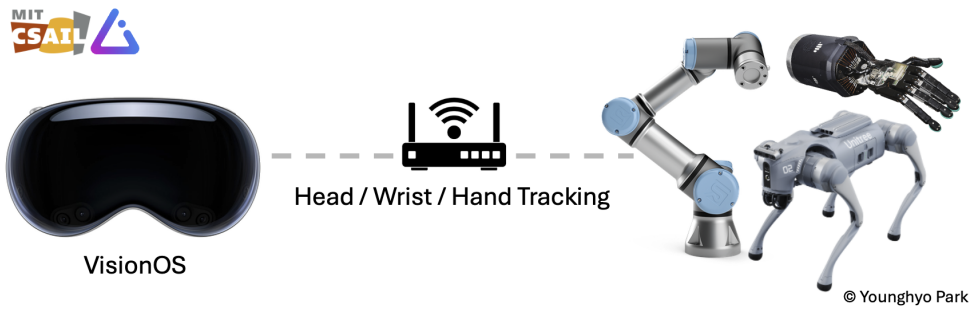


Figure 4.1: The app streams human movements to robots connected to the same network.

4.2 Tracking Streamer: VisionOS App

Tracking Streamer is an VisionOS app that you can install from the App Store. The code is also open-sourced in the repository. The app serves two primary purposes: (a) tracking the human movements, and (b) streaming the tracking data over network.

Retrieving Tracking Data The app uses Apple's ARKit as its core library to track every movements of the user wearing AVP. There are three main movements that are being tracked by the ARKit: (a) head, (b) wrist, and (c) fingers. Note that AVP also tracks user's eye movements, but Apple restricts developer's direct access to the tracking data due to privacy concerns.

To get the device's location (head frame), the app constantly calls `queryDeviceAnchor` which returns the SE(3) location of the device with respect to a fixed global frame $\mathbf{T}_g$. Global frame is initialized when the app is first launched, and is attached to a ground where you're in. You can also reset the global frame to current position by long-pressing the digital crown. The head frame (device location) is then tracked using Apple's own localization algorithm, which can accurately locate the user even in large scale environments (Video). The app also uses `HandTrackingProvider` to track the user's wrist and fingers during the session. It tracks the position and orientation of user's two wrists (left and right) with respect to the ground frame, i.e., $\mathbf{T}_{\text{gw}_\text{L}}, \mathbf{T}_{\text{gw}_\text{R}}$, and tracks the pose of 25 finger joints in each hand with respect to its wrist frame, i.e., $\mathbf{T}^i_{\text{wf}}$.

Communication with gRPC The app then uses gRPC as its network communication protocol to stream the data to any clients existing in the same network. Use of gRPC instead of other robotics-oriented communication protocol (e.g. ROS2) allows the data to be subscribed from wider range of devices including Linux, Mac, Windows machines.

4.3 Python API

Subscribing to the data is easy: users can simply install the Python package:

```
pip install avp_stream
```

Below code snippet demonstrates how developers can access the data it's being streamed.

```
from avp_stream import VisionProStreamer
avp_ip = "10.31.181.201"   # example IP
s = VisionProStreamer(ip = avp_ip, record = True)

while True:
    r = s.latest  # gets the latest tracking data
    print(r)
```

4.3.1 Available Data

The tracking data is represented as a dictionary containing the following key/values. Most of them are raw data streamed from the device, but some are post-processed by the Python library.

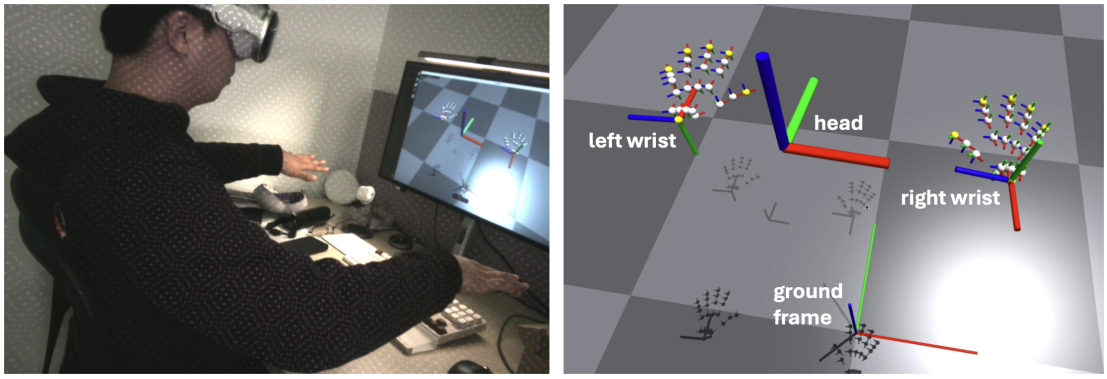


Figure 4.2: How the axes are defined for head, wrist, and fingers.

- `right/left_wrist`: SE(3) pose of right/left wrist, measured from ground frame. `np.array` with shape (1,4,4)
- `right/left_fingers`: SE(3) pose of 25 finger joints in right/left hand, measured from the wrist frame. `np.array` with shape (25,4,4).
- `head`: SE(3) pose of the head, measured from the ground frame. `np.array` with shape (1,4,4).
- `right/left_pinch_distance`: distance between thumb and index finger. `float`.
- `right/left_wrist_roll`: roll rotation of your wrist. Measures how you rotate your wrist around your arm axis. `float`.

4.3.2 Axes Conventions

Figure 4.2 shows how the axes are defined for each components. Note that Apple originally uses `Y_AXIS_UP` convention for their tracking. Considering that most robotics applications use `Z_AXIS_UP` instead, our python library automatically converts Apple's representation following `Z_AXIS_UP` convention.

4.3.3 Recording Data

If you pass in `record = True` when initializing the `VisionProStreamer` class, it will simply store every streamed data into a list. To access and save the recorded data, simply call:

```
# save the recording when the app finishes
torch.save(s.recording, file_path)
```

4.4 Things to be careful about

During our experiments, we realized couple of scenarios where the app didn't behave as we expected. Most of these failure cases actually stemmed from our misunderstanding of how Apple's ARKit is designed to work.

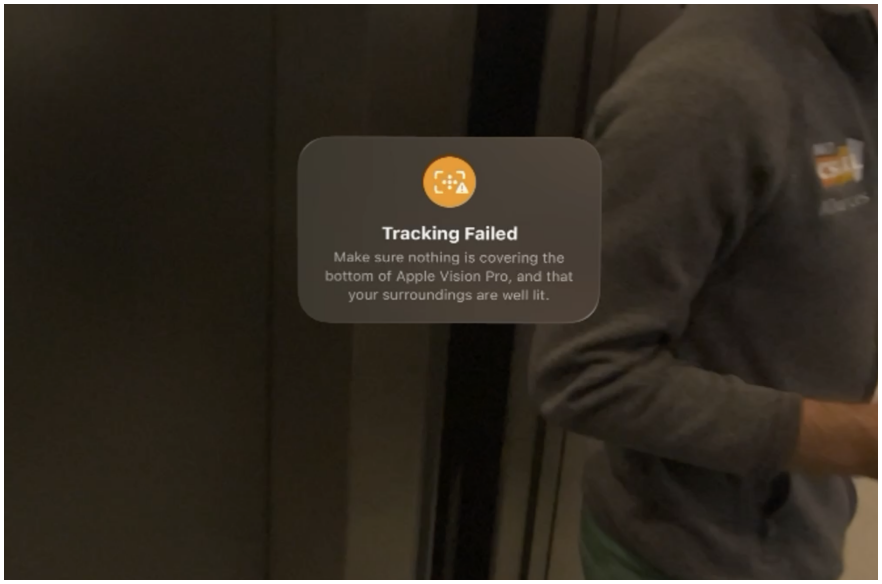


Figure 4.3: AVP fails to track its device location whenever it's in a moving vehicle (e.g. elevators, airplanes, cars, trains).

- **Don't use it inside an elevator (or any moving vehicles!)** — Apple's ARKit will fail to localize the device while you're in a contained, but moving, environment. That includes airplanes, cars, and even elevators! That's why AVP has a separate "airplane" mode which you can turn it on from control center (perhaps a mode that exclusively uses visual information for localization). Whenever you're in those environments without the mode being turned on, you'll see this error message (Figure 4.3) and every tracking will stop.
- **If you're moving downhill, your z will also be decreasing.** — Once you opened the app, the ground will not move. It'll stay there unless you actively reintialize your ground frame by long-pressing the digital crown. If you're moving slightly downhill (or uphill), for instance, your **z** will actually decrease (and increase) without you noticing. This might actually affect the robot's behavior, depending on how you designed to use the tracking.
- **If you put your hands completely down, AVP cannot see your hands.** — When you're walking normally with your hands around your waist/hip, AVP struggles to detect your hands. In those scenarios, the hand tracking stream might be a bit noisy. The library doesn't do any smoothing or filtering on the streamed data: so you might have to implement your own post-processing function to filter out those noisy estimations.

Chapter 5

DART: Robot Data Collection in Virtual Reality

5.1 Introduction

A major bottleneck in building a generalist robot is the lack of diverse and high-quality data appropriate for training robust and generalizable sensorimotor control policies. The dream is an internet-scale robotics dataset that continually and rapidly grows with data uploaded from around the world — just like many people upload language, images, and videos online. Despite many recent efforts [3–6], we are not there yet. In this paper, we examine and address many key bottlenecks in achieving this dream.

Consider collecting data to perform a task, such as moving dishes from the sink to the dishwasher. The data collector's first challenge is ***setting up the environment***. There are two options: physically construct a kitchen in the lab around the robot or physically move the robot to an actual kitchen. Neither is easy to scale as data from many kitchens will be needed.

Once the environment is set up, operating the robot leads to the second challenge – ***observing and understanding*** the scene. For instance, due to visual occlusions and lack of tactile feedback, operators may be unable to sense an object's motion resulting from the robot's action. Further, if the teleoperation is remote, it adds additional challenges originating from network delays, limited field of view, and visual artifacts. Such challenges can slow down operators and often prevent them from performing dynamic or precise tasks.

If the data collector succeeds at resolving the first two challenges and moves all the dishes from the sink to the dishwasher to complete the exemplar task, a third obstacle emerges: all the dishes must be returned to the sink to collect a new trajectory! In addition to being time-consuming, this ***resetting*** is physically and mentally exhausting as operators must context-switch between robot control and environment setup. Ensuring that each reset presents the robot with a diverse range of scenarios is also mentally taxing requiring imagination.

What makes the experience even worse for the operators is the need to ***repeat*** the process of *teleoperating* and *resetting* a large number of times. The number of required demonstrations and the need for diversity in demonstrations scales with the task complexity and the extent

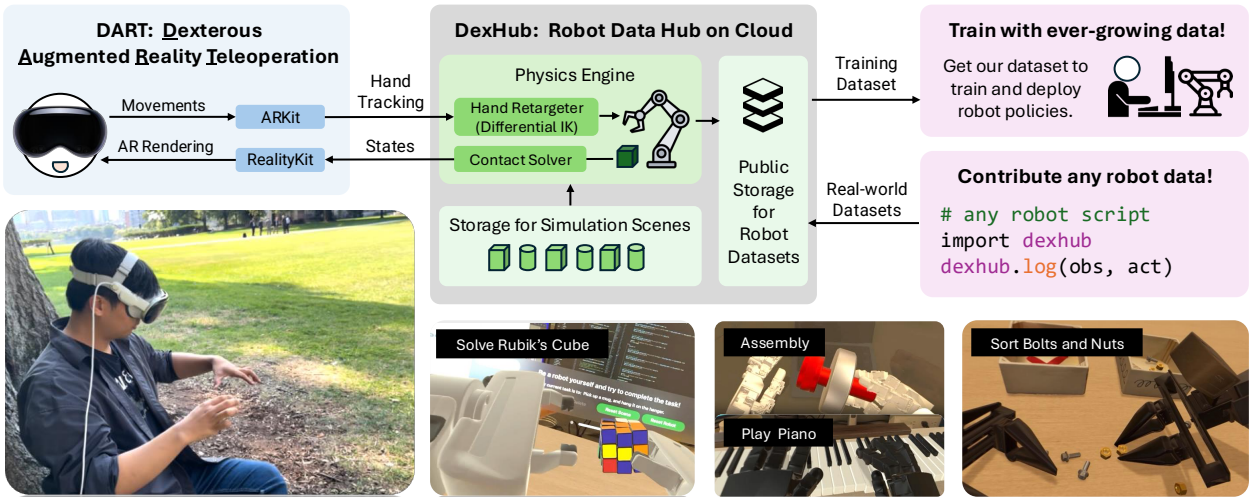


Figure 5.1: We present **DART**, Dexterous Augmented Reality Teleoperation system, enabling intuitive, low-latency teleoperation with cloud-hosted simulation. Through a user study, we found that DART enables first-time robot teleoperators to achieve 2.1× faster data collection throughput with significantly lower physical fatigue than existing real-world teleoperation platforms. To further support scaling up data collection efforts in the community, we also release , a cloud-hosted data hub for robot learning where data collected in DART is automatically stored. https://dexhub.ai/project

of required generalization. Unfortunately, humans are known to lose focus when performing a repetitive job [77].

Finally, say the operator has finished collecting a few hundred demonstrations. How does the recorded data get ***processed, stored, and used***? It is common to store collected demonstrations on a local machine or a *private* cloud, which is often not shared to general publicly unless someone explicitly requests it.

These issues, all combined, make existing data collection methods struggle to scale up without operator fatigue. Making everything worse, the data collected in real-world has limited applicability in terms of policy training methods; reinforcement learning, for instance, cannot be easily applied on top of real-world demonstrations as it lacks a digital twin where virtual agents can freely explore and self improve its performance. A data collection method that (a) can be easily parallelized and crowd-sourced with minimal hardware requirements and (b) wide range of policy training pipelines can be applied can get closer to the needed scale and diversity of robot data.

To that end, we introduce **DART**, a Dexterous Augmented Reality Teleoperation system, enabling *anyone* in the world to teleoperate robots in simulation with an intuitive, game-like AR interface. Connected to a cloud-hosted simulation, DART allows users to collect demonstrations for an unlimited number of scenes in one sitting without having to physically set up environments or physically move robots to different places. DART's high-fidelity AR rendering allows users to observe the scene in great detail with minimal occlusion, enabling teleoperation of complex tasks. DART also allows users to reset the environment with a click of a button, removing the taxing process of physically resetting the scene.

As a result, our user study shows that DART achieves 2.1× **faster data collection throughput** with significantly less physical and cognitive fatigue on tasks requiring fine-grained control compared to most existing robot data collection pipelines. Our experiments also highlight the unmatched benefits of collecting demonstrations in simulation over the real world. Simulation-trained policies achieve higher robustness than real-world trained policies due to data augmentation and randomization strategies only possible in simulation. Finally, all robot demonstrations collected through DART are automatically stored and logged to our public cloud-hosted database, , which serves as an open-sourced data hub for robot learning.

Our key contributions are outlined as follows:

1. In Sec. 5.2, we introduce DART, a novel AR-based teleoperation platform, and detail its system architecture and supported features. We also showcase the diversity of tasks we can perform with DART, unlocked by enhanced teleoperation experience.

2. In Sec. 5.3.1, we analyze the impact of different teleoperation interface design choices through user study. We show that DART enables higher data collection throughput and lower fatigue than alternatives.

3. In Sec. 5.3.2, we show that policies trained with data collected via DART can be effectively transferred to the real world and are more robust than those trained with real-world demos.

5.2 System Design

This section details the system architecture of DART and its benefits (Sec 5.2.1). We then introduce the main features of the platform (Sec 5.2.2), which are designed to maximize the platform's capability (Sec 5.2.3) and enhance user experience.

5.2.1 System Architecture

DART's key components facilitate intuitive, low-latency teleoperation available for anyone worldwide.

Simulation Assets as AR Objects

Enabled by Apple's RealityKit, DART presents all assets in simulation environments, including robots, as photo-realistic AR objects overlayed over each operator's real-world environment. Handling visualization locally on the AR device (a) removes unnecessary latency from transmitting large image data packets and (b) significantly improves the real-timeliness of the simulation by removing the compute-intensive rendering layer. Variation in latency critically impacts the user's data collection throughput and cognitive fatigue, as highlighted by our user study (See Sec. 5.3.1).

		DART (Ours)	Others
Human → Robot	Data Type	Hand Tracking	Hand and Head Tracking
		25 Hand Keypoints × SE(3)	(25 Hand Keypoints + 1 Head) × SE(3)
	Packet Size	0.7kB	0.728kB
Robot → Human	Data Type	Oracle Sim States	Stereo RGB image
		n joints × `float` m objects × SE(3)	2×(480×640×3) `uint8`
	Packet Size	**1.6kB**	**1843.2kB**

Table 5.1: We highlight DART's **1,000× reduction** in network packet size between robot and operator's AR device compared to existing frameworks. $n = 58$, $m = 50$ assumed for DART.

Low-Latency Communication

Communication between the AR device, i.e., Apple Vision Pro, and the cloud-hosted simulation is handled via gRPC, which facilitates low-latency, *asynchronous* bidirectional data transfer. The AR device sends hand-tracking data to the simulation, and asynchronously receives the simulation state. Table 5.1 highlights the reduced network load of our approach compared to a typical setting where real-world or simulated camera streams are transmitted over the network. Even in the most adversarial case, where robots have $n = 58$ joints and simulation scenes contain $m = 50$ objects, the data packet size is over 1,000× smaller than that required for existing teleoperation frameworks.

Cloud-Hosted Simulation

The robot simulation is powered by MuJoCo [78] and dynamically launched on AWS Elastic Container Registry (ECR) as users join. Each simulation instance runs in the cloud, enabling open access and low user setup costs. Due to compact packet sizes (Table 5.1), cloud-hosting does not critically impact the overall latency of our platform compared to local-hosting, as evidenced in Table 5.2.

Hand Tracking and Mapping

DART leverages Apple's ARKit to track poses of hand and wrist keypoints. We use a subset of detected keypoints, which fully determine the end-effector and finger movements, as target points for robots to track. Specifically, for robot systems with parallel-jaw grippers, we use the xyz position of 4 finger key points as tracking targets, which fully determine the SE(3) pose of the robot's end-effector (Fig. 5.2). DART uses differential inverse kinematics [79] by defining position-only tracking costs for each keypoints, $e(\mathbf{p})$. We additionally apply basic safety constraints, i.e., self-collision avoidance, expressed as $d(q)$. The resulting optimization

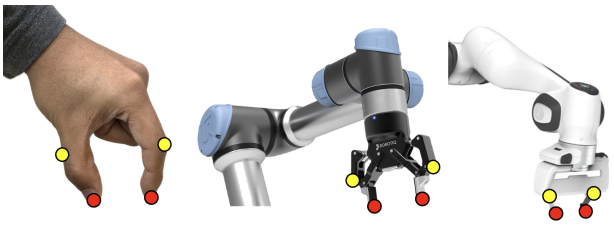


Figure 5.2: 4 finger keypoints used as tracking points for robots with parallel-jaw grippers.

	Cloud-Hosted Simulation	Local Machine and Local Network
CPU	AWS EC2 C7i	i9-13900k
Packet Travel Time	15.4 ms	10.3 ms
Simulation Step	1.8 ms	1.6 ms
Total	17.2 ms	11.9 ms

Table 5.2: Comparing the time profile of our system running on the cloud v/s hosted on a local machine. AWS instance was hosted on us-east-1, connected from Boston.

problem is as follows,

$$\begin{aligned} &\min_{v \in c} \sum_{\mathbf{p} \in \mathcal{P}} \|J_e(q)v + \alpha e(\mathbf{p})\|^2 \\ &\text{s.t. } v_{\min}(q) \leq v \leq v_{\max}(q),\ d(q) > 0. \end{aligned} \tag{5.1}$$

For dexterous five-fingered hands, we use six position-only keypoints – five from the fingertips and one from the wrist.

5.2.2 System Features

DART supports a wide range of features to enhance the teleoperation experience while maintaining low setup costs, allowing *anyone* to participate in robotics data collection. Although it is currently developed for Apple Vision Pro, Apple's AR device, the design decisions can also be developed and applied for different AR devices.

Multiple Robots and Tasks

Out-of-the-box, DART supports many robots: multiple end-effectors (Robotiq 2F-85 gripper, Panda Hand, Allegro Hand) can be attached to bimanual setup of Franka Research 3. Unitree G1 and ALOHA [80] are also supported. High-fidelity MuJoCo models of these robots were provided by [81].

One-Click Reset

DART includes an efficient task-resetting feature in simulation. Users can reset the environment with a single click of a button, significantly reducing operator fatigue and increasing data collection throughput.

Instant Task Switching

In addition to resetting a single scene, DART enables quick switching between various tasks and simulation environments. This functionality minimizes the operator's mental fatigue that arises from repetitively performing the same task, allowing for a more engaging data collection experience.

5.2.3 Capability and Task Diversity

DART is capable and versatile. It supports a wide range of tasks, from simple object manipulation to complex, precise, and dexterous maneuvers, as highlighted in Figure 5.1. These examples and those below illustrate the platform's potential to support various research and practical applications in robotics.

- Fine motor skills: e.g., picking up small objects.
- Household chores: e.g., hanging mugs on a rack.
- Dexterous Manipulation: e.g., solving a Rubik's cube.

5.3 Experiments

Our experiments address two key questions:

1. *How **intuitive** is DART for robotics novices to use?* We conduct a formal user study to assess the platform's accessibility to individuals without robotics expertise. (Section 5.3.1)
2. *Can the data collected in simulation be effectively **transferred to real-world** robots?* We demonstrate that policies trained on data collected through DART transfer zero-shot to real environments with simple Sim2Real techniques. We also highlight the generalizability of DART policies compared to those trained with real-world data. (Section 5.3.2)

5.3.1 User Study

Through a controlled user study, we analyze the impact of DART's design decisions on intuitiveness and usability. Specifically, we compare: (a) the experience of collecting data in real-world versus simulation environments (Sec 5.3.1), (b) methods of visual perception (Sec 5.3.1), and (c) control interfaces (Sec 5.3.1). A total of nine participants with no prior experience in robotics were recruited.

In varying settings, participants spent 7 minutes collecting as many robot demonstrations as possible. We asked the participants to organize 10 bolts and nuts from a table into boxes. Participants were responsible for resetting the scene both in simulation and real-world environments via reset button or manual effort, respectively. Participants teleoperated two ViperX arms with parallel-jaw grippers, and kinematically equivalent teacher devices were

	DART	Modulation of **Command Interface**	Modulation of **Visual Feedback Design**			ALOHA [80]
		Finger Tracking ↓ Kinematic Double	Rendering as AR Objects ↓ Sim Rendering (RGB, Stereo)	Rendering as AR Objects ↓ Sim Rendering (RGB, Mono)	Active Viewpoint ↓ Fixed Viewpoint	
Data Throughput	**7.8** parts / min	6.8 parts / min	3.6 parts / min	3.0 parts / min	2.7 parts / min	**3.7** parts / min

Table 5.3: Quantitative comparison between different teleoperation setups for two ViperX arms with parallel-jaw gripper [80]. Users are tasked to organize ten bolts and nuts into two boxes, and DART allowed users to organize 7.77 parts per minute on average, while modulation of both command interface and visual feedback settings dropped the performance significantly. We report percent change in throughput relative to DART averaged across users.

used as a real-world teleoperation interface [80]. Quantitative results are presented in Table 5.3; further analysis follows.

Teleoperating in Real-World vs Simulation

Our user study comparing DART and real-world teleoperation revealed two key findings. First, a significant portion of time in real-world data collection is spent physically resetting the environment and managing unexpected hardware failures (e.g., performing electrical resets after motor malfunctions) as reported in Fig. 5.4. By contrast, most of the time in DART is dedicated to actual data collection.

Second, even after accounting for reset times and hardware malfunctions, participants in real-world teleoperation showed around **2**× lower data collection throughput. For a comparison experiment with wide range of real-world data collection systems, we used two different robot systems: dual ViperX arms [80] and RB-Y1 from Rainbow Robotics. Both data collection system has kinematic double as its teleoperation interface. Total 20 participants were asked to perform 4 bimanual tasks ranging from relatively simple object rearrangment task to precise insertion tasks. Figure 5.3 shows the data throughput comparison between DART and two different real-world robot systems on four different tasks.

Many participants attributed this considerable data throughput gap to a) physical fatigue during teleoperation and b) their inability to closely observe local contact interactions, which hindered their ability to perform tasks effectively (Table 5.3). This particular attribution becomes evident with following ablation studies.

Effect of Visual Observation on Human Operator's Performance

Our key findings are threefold. First, transmitting images over a network inevitably introduces a tradeoff between latency and decreased visual fidelity, which can negatively impact teleoperation experience. All methods transmitting simulation renderings over the network (those with stero and mono rendering) suffered a significant drop in user's data collection throughput compared to DART which transmits only the raw simulation states.

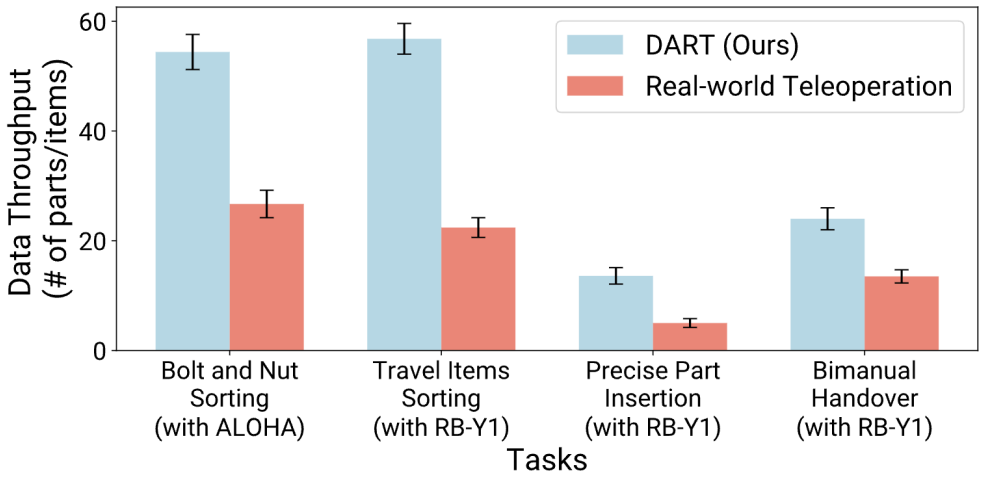


Figure 5.3: Data throughput comparison between DART and real-world teleoperation systems. For each robot and task, five participants were asked to teleoperate the tasks as many as possible for 7 minutes. For real-world teleoperation, kinematically equivalent teacher device, i.e., kinematic double, was used as a teleoperation interface.

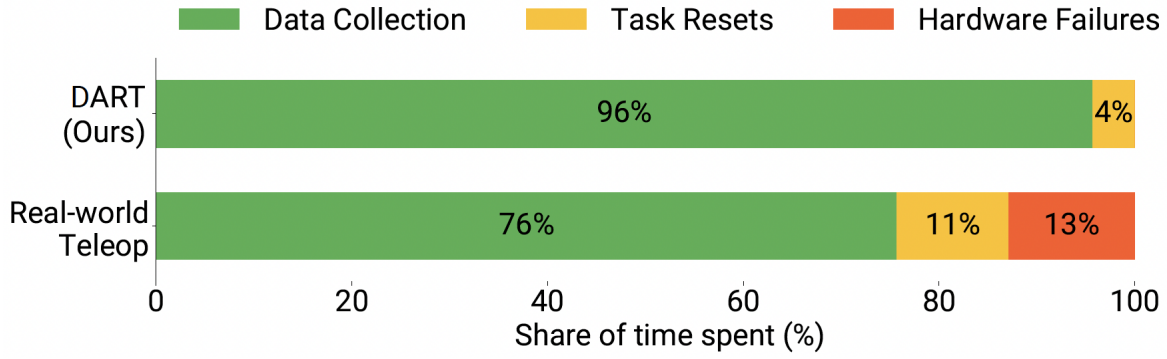


Figure 5.4: DART allows operators to spend more time on actual data collection, rather than supplementary tasks such as resetting the environment for every task completion or dealing with hardware failures.

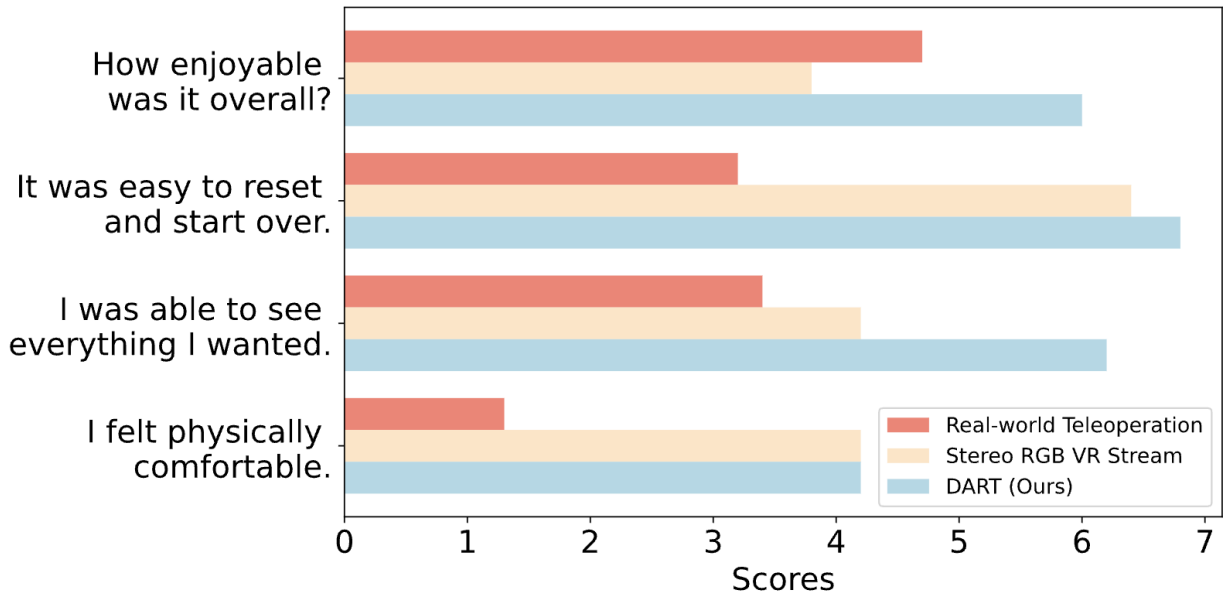


Figure 5.5: Qualitative comparison between different teleoperation interfaces amongst user study participants. Participants reported that DART is enjoyable, physiaclly less fatiguing and allows better visual observation during teleoperation.

Second, we find that mono rendering, which limits the ability to properly perceive depth, suffered a performance drop over stereo rendering. Additionally, some participants reported feeling nauseous (Table 5.3) with stereo rendering – which uses a fixed interpupillary distance (IPD). By contrast, DART relies on VisionOS's [1] native rendering engine, which dynamically adjusts to each user's IPD [82].

Finally, we found that active perception, where users can explore their surroundings and adjust their viewpoint by moving their heads, is critical. Teleoperation without active perception reduces the data collection rate by 21.7%.

Control Method

We compared two methods for operating robots in simulation: a) a kinematically equivalent teacher device and b) inverse kinematics (IK) using hand tracking keypoints as targets. Our findings indicate that the kinematic double did not significantly improve task success rate over its IK equivalent. While the kinematic double provides more direct control over the robot's joints, users reported that the intuitive hand tracking offered by DART was sufficient, or even better, due to reduced weight and strain on the operator (Table 5.3).

5.3.2 Sim2Real and Generalizability

Both DART and real-world data collection offer distinct advantages for real-world policy training. With DART, roboticists benefit from significantly higher data throughput with

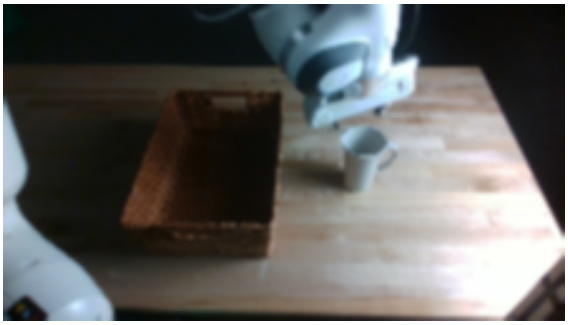
[1] Apple's Operating System for AR devices

(a) Nominal Lab Setting

(b) Camera Pose Change

(c) Unseen Distractions

(d) Green background



(e) Lighting Change

(f) Location Change

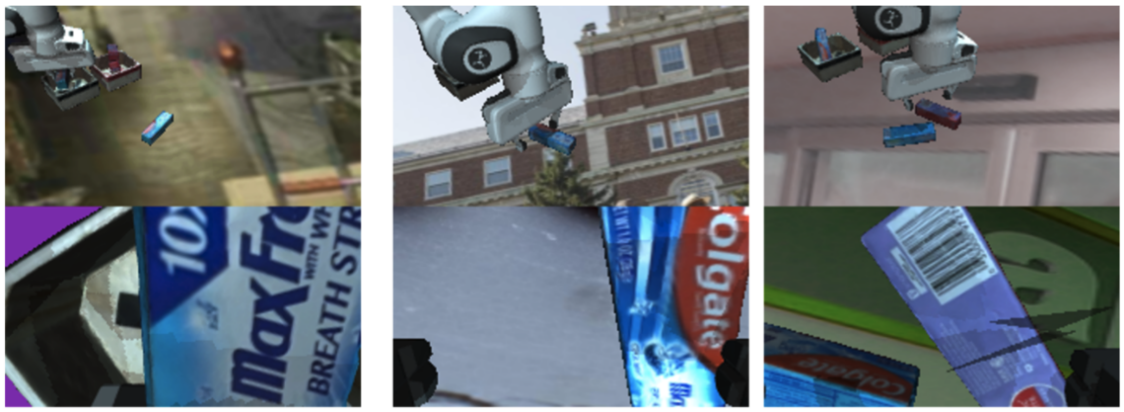
Figure 5.6: Six different settings to evaluate the robustness of our RGB vision-based policy trained with data collected through DART.

reduced physical and cognitive demands, as demonstrated by our user study (Sec. 5.3.1). One minor downside of using DART is the need to import scenes into the simulation environment. Fortunately, with modern advances in computer vision [83, 84], scanning 3D objects from the real world has become incredibly efficient. The bigger challenge, however, lies in bridging the potentially large Sim2Real gap. Given these trade-offs, how does one weigh the benefits of faster data collection against the challenge of real-world deployment?

Our experimental results suggest that collecting data in simulation offers **more advantages than drawbacks** when paired with a proper Sim2Real pipeline. In particular, we demonstrate the unique robustness of Sim2Real-transferred policies, enabled by diverse data augmentation techniques only available in simulation environments.

Specifically, we compare two types of RGB vision policies: (a) a policy trained on real-world data, and (b) a policy trained on simulation data collected through DART. Both policies are trained on two tasks with 50 minutes of operator effort. Both policies also use

(a) Real-world Images



(b) Simulation Renderings with Background Augmentations

Figure 5.7: Visual comparison between training images for Real and DART policies. Simulation allows augmentation out-of-the-box, which results in zero-shot Sim2Real and robustness.

a standard ACT [80] implementation at 20Hz. Real-world datasets are augmented with Gaussian blur and color-jitter. DART datasets were additionally augmented by randomizing the camera extrinsic and intrinsics, replacing the background with random textures and images from [85–87], and randomizing the lighting setting in simulation (Figure 5.7).

Inspired by [88], we evaluated policies in six diverse environments in the real world illustrated in Fig. 5.6. We found that our DART policies not only demonstrate zero-shot Sim2Real in the nominal setting but also significantly outperform the Real policy in many of the modified settings (Table 5.4). Our results highlight the benefit of scaling up simulation data versus real-world data: a single demo in simulation, which can be aggressively augmented, is more valuable for learning than that collected in real world.

Task	Pick Mug in Basket		Sorting Small Items	
Trained on data from	Real-world	DART	Real-world	DART
Lab Space	65%	**80%**	45%	**60%**
with:				
Lighting Changes	45%	**45%**	25%	**65%**
Background Changes	10%	**60%**	5%	**40%**
Cam. Pose Changes	5%	**35%**	0%	**40%**
Unseen Distractions	0%	**70%**	0%	**45%**
Communal Kitchen	0%	**50%**	0%	**35%**

Table 5.4: Success rates for policies trained with 50 minutes of data collection effort in the real-world v/s DART. The results highlight the robustness of policies trained with simulation data, enabled by diverse data augmentation strategies.

Chapter 6

Concluding Remarks

6.1 Summary of Contributions

This thesis explored two critical bottlenecks in scaling up robot learning: the difficulty of environment design in reinforcement learning (RL) and the dependence on physical hardware for robot data collection. First, we identified that much of the human effort in RL does not go into algorithm tuning, but into environment shaping—designing the rewards, observations, action spaces, and task dynamics to make learning feasible. We formalized this challenge and proposed a roadmap toward automating environment shaping, which we argue is essential for generalizing RL across tasks and robot morphologies.

Second, we developed new tools for robot data collection without real robots. DART, an augmented reality teleoperation system using the Apple Vision Pro, allows users to intuitively collect high-quality demonstrations in simulation. Our user studies showed that DART achieves over 2× the throughput of real-world teleoperation systems while reducing physical and cognitive fatigue.

6.2 Limitations

Despite these contributions, important limitations remain. For RL, the process of environment shaping is still heavily task-dependent and challenging to generalize. Automating shaping across diverse robot morphologies and task types remains an open research problem. While we proposed benchmarks and tools to support this direction, a fully automatic shaping framework remains future work.

In the realm of data collection, systems like DART help mitigate physical hardware bottlenecks but still rely on simulation environments. The fidelity of physics engines, especially for complex contact dynamics, deformable objects, and sensor modeling, remains a limiting factor for real-world deployment. Furthermore, while we demonstrated successful Sim2Real transfer in limited settings, general-purpose transfer remains unsolved.

6.3 Opportunities and Future Directions

Several promising directions emerge from this work. Automating environment shaping could be approached as a meta-learning or program synthesis problem, where shaping functions are learned from prior tasks. On the data side, the continued integration of high-fidelity AR/VR hardware with cloud simulation infrastructure enables new interfaces for data collection and model deployment. With wider adoption of wearable and spatial computing devices, large-scale, crowd-sourced robot datasets may become viable—mirroring the trajectory of data collection in NLP and computer vision.

Moreover, combining alternative data sources (e.g., human video and simulation) under a shared representation may unlock new training paradigms. Addressing the Sim2Real and morphology gaps will require novel alignment techniques and more realistic simulation capabilities.

References

[1] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, et al. "Rt-1: Robotics transformer for real-world control at scale". In: *arXiv preprint arXiv:2212.06817* (2022).

[2] H. Walke et al. "BridgeData V2: A Dataset for Robot Learning at Scale". In: *Conference on Robot Learning (CoRL)*. 2023.

[3] A. Khazatsky et al. "DROID: A Large-Scale In-The-Wild Robot Manipulation Dataset". In: (2024).

[4] A. O'Neill, A. Rehman, A. Maddukuri, A. Gupta, A. Padalkar, A. Lee, A. Pooley, A. Gupta, A. Mandlekar, A. Jain, et al. "Open X-Embodiment: Robotic Learning Datasets and RT-X Models: Open X-Embodiment Collaboration 0". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 6892–6903.

[5] H.-S. Fang, H. Fang, Z. Tang, J. Liu, C. Wang, J. Wang, H. Zhu, and C. Lu. "Rh20t: A comprehensive robotic dataset for learning diverse skills in one-shot". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 653–660.

[6] F. Ebert, Y. Yang, K. Schmeckpeper, B. Bucher, G. Georgakis, K. Daniilidis, C. Finn, and S. Levine. "Bridge data: Boosting generalization of robotic skills with cross-domain datasets". In: *arXiv preprint arXiv:2109.13396* (2021).

[7] V. Makoviychuk et al. *Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning*. 2021.

[8] S. Ross and D. Bagnell. "Efficient Reductions for Imitation Learning". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Y. W. Teh and M. Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 661–668. URL: https://proceedings.mlr.press/v9/ross10a.html.

[9] S. Ross, G. Gordon, and D. Bagnell. "A reduction of imitation learning and structured prediction to no-regret online learning". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2011, pp. 627–635.

[10] T. Z. Zhao, J. Tompson, D. Driess, P. Florence, S. K. S. Ghasemipour, C. Finn, and A. Wahid. "ALOHA Unleashed: A Simple Recipe for Robot Dexterity". In: *8th Annual Conference on Robot Learning*.

[11] L. Ankile, A. Simeonov, I. Shenfeld, M. Torne, and P. Agrawal. "From Imitation to Refinement–Residual RL for Precise Visual Assembly". In: *arXiv preprint arXiv:2407.16677* (2024).

[12] Y. Qin, W. Yang, B. Huang, K. Van Wyk, H. Su, X. Wang, Y.-W. Chao, and D. Fox. "Anyteleop: A general vision-based dexterous robot arm-hand teleoperation system". In: *arXiv preprint arXiv:2307.04577* (2023).

[13] M. Mosbach, K. Moraw, and S. Behnke. "Accelerating interactive human-like manipulation learning with gpu-based simulation and high-quality demonstrations". In: *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*. IEEE. 2022, pp. 435–441.

[14] A. Iyer, Z. Peng, Y. Dai, I. Guzey, S. Haldar, S. Chintala, and L. Pinto. "Open teach: A versatile teleoperation system for robotic manipulation". In: *arXiv preprint arXiv:2403.07870* (2024).

[15] X. Cheng, J. Li, S. Yang, G. Yang, and X. Wang. "Open-TeleVision: teleoperation with immersive active visual feedback". In: *arXiv preprint arXiv:2407.01512* (2024).

[16] J. Duan, Y. R. Wang, M. Shridhar, D. Fox, and R. Krishna. "Ar2-d2: Training a robot without a robot". In: *arXiv preprint arXiv:2306.13818* (2023).

[17] S. Chen, C. Wang, K. Nguyen, L. Fei-Fei, and C. K. Liu. "ARCap: Collecting High-quality Human Demonstrations for Robot Learning with Augmented Reality Feedback". In: *arXiv preprint arXiv:2410.08464* (2024).

[18] J. van Haastregt, M. C. Welle, Y. Zhang, and D. Kragic. "Puppeteer Your Robot: Augmented Reality Leader-Follower Teleoperation". In: *arXiv preprint arXiv:2407.11741* (2024).

[19] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. "Learning fine-grained bimanual manipulation with low-cost hardware". In: *arXiv preprint arXiv:2304.13705* (2023).

[20] Z. Fu, T. Z. Zhao, and C. Finn. "Mobile ALOHA: Learning Bimanual Mobile Manipulation with Low-Cost Whole-Body Teleoperation". In: *arXiv preprint arXiv:2401.02117* (2024).

[21] J. Luo, Z. Hu, C. Xu, Y. L. Tan, J. Berg, A. Sharma, S. Schaal, C. Finn, A. Gupta, and S. Levine. "SERL: A Software Suite for Sample-Efficient Robotic Reinforcement Learning". In: *arXiv preprint arXiv:2401.16013* (2024).

[22] P. Wu, A. Escontrela, D. Hafner, P. Abbeel, and K. Goldberg. "Daydreamer: World models for physical robot learning". In: *Conference on Robot Learning*. PMLR. 2023, pp. 2226–2240.

[23] K. Lei, Z. He, C. Lu, K. Hu, Y. Gao, and H. Xu. "Uni-O4: Unifying Online and Offline Deep Reinforcement Learning with Multi-Step On-Policy Optimization". In: *arXiv preprint arXiv:2311.03351* (2023).

[24] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. "Learning robust perceptive locomotion for quadrupedal robots in the wild". In: *Science Robotics* 7.62 (2022), eabk2822.

[25] Y. Ji, G. B. Margolis, and P. Agrawal. "DribbleBot: Dynamic Legged Manipulation in the Wild". In: *International Conference on Robotics and Automation* (2023).

[26] D. Hoeller, N. Rudin, D. Sako, and M. Hutter. "ANYmal Parkour: Learning Agile Navigation for Quadrupedal Robots". In: *arXiv preprint arXiv:2306.14874* (2023).

[27] Z. Zhuang, Z. Fu, J. Wang, C. Atkeson, S. Schwertfeger, C. Finn, and H. Zhao. "Robot parkour learning". In: *arXiv preprint arXiv:2309.05665* (2023).

[28] X. Cheng, K. Shi, A. Agarwal, and D. Pathak. "Extreme parkour with legged robots". In: *arXiv preprint arXiv:2309.14341* (2023).

[29] F. Jenelten, J. He, F. Farshidian, and M. Hutter. "DTC: Deep Tracking Control". In: *Science Robotics* 9.86 (2024), eadh5401.

[30] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al. "Learning dexterous in-hand manipulation". In: *The International Journal of Robotics Research* 39.1 (2020), pp. 3–20.

[31] T. Chen, M. Tippur, S. Wu, V. Kumar, E. Adelson, and P. Agrawal. "Visual dexterity: In-hand reorientation of novel and complex object shapes". In: *Science Robotics* 8.84 (2023), eadc9244.

[32] A. Handa, A. Allshire, V. Makoviychuk, A. Petrenko, R. Singh, J. Liu, D. Makoviichuk, K. Van Wyk, A. Zhurkevich, B. Sundaralingam, et al. "Dextreme: Transfer of agile in-hand manipulation from simulation to reality". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 5977–5984.

[33] Z.-H. Yin, B. Huang, Y. Qin, Q. Chen, and X. Wang. "Rotating without Seeing: Towards In-hand Dexterity through Touch". In: *arXiv preprint arXiv:2303.10880* (2023).

[34] Y. Song, A. Romero, M. Müller, V. Koltun, and D. Scaramuzza. "Reaching the limit in autonomous racing: Optimal control versus reinforcement learning". In: *Science Robotics* 8.82 (2023), eadg1462.

[35] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza. "Champion-level drone racing using deep reinforcement learning". In: *Nature* 620.7976 (2023), pp. 982–987.

[36] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[37] H. Wei, G. Zheng, H. Yao, and Z. Li. "Intellilight: A reinforcement learning approach for intelligent traffic light control". In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2018, pp. 2496–2505.

[38] C. Wu, A. Kreidieh, E. Vinitsky, and A. M. Bayen. "Emergent behaviors in mixed-autonomy traffic". In: *Conference on Robot Learning*. PMLR. 2017, pp. 398–407.

[39] X.-Y. Liu, J. Rui, J. Gao, L. Yang, H. Yang, Z. Wang, C. D. Wang, and G. Jian. "FinRL-Meta: Data-Driven Deep ReinforcementLearning in Quantitative Finance". In: *Data-Centric AI Workshop, NeurIPS* (2021).

[40] J. R. Vázquez-Canteli, J. Kämpf, G. Henze, and Z. Nagy. "CityLearn v1. 0: An OpenAI gym environment for demand response with deep reinforcement learning". In: *Proceedings of the 6th ACM international conference on systems for energy-efficient buildings, cities, and transportation*. 2019, pp. 356–357.

[41] J. D. Co-Reyes, S. Sanjeev, G. Berseth, A. Gupta, and S. Levine. "Ecological reinforcement learning". In: *arXiv preprint arXiv:2006.12478* (2020).

[42] O. G. Selfridge, R. S. Sutton, and A. G. Barto. "Training and Tracking in Robotics." In: *Ijcai*. 1985, pp. 670–672.

[43] A. Y. Ng, D. Harada, and S. Russell. "Policy invariance under reward transformations: Theory and application to reward shaping". In: *Icml*. Vol. 99. Citeseer. 1999, pp. 278–287.

[44] S. Singh, R. L. Lewis, and A. G. Barto. "Where do rewards come from". In: *Proceedings of the annual conference of the cognitive science society*. Cognitive Science Society. 2009, pp. 2601–2606.

[45] J. Eßer, N. Bach, C. Jestel, O. Urbann, and S. Kerner. "Guided Reinforcement Learning: A Review and Evaluation for Efficient and Effective Real-World Robotics [Survey]". In: *IEEE Robotics & Automation Magazine* 30.2 (2023), pp. 67–85. DOI: 10.1109/MRA.2022.3207664.

[46] P. Agrawal. "The task specification problem". In: *Conference on Robot Learning*. PMLR. 2022, pp. 1745–1751.

[47] X. B. Peng and M. Van De Panne. "Learning locomotion skills using deeprl: Does the choice of action space matter?" In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 2017, pp. 1–13.

[48] E. Aljalbout, F. Frank, M. Karl, and P. van der Smagt. "On the Role of the Action Space in Robot Manipulation Learning and Sim-to-Real Transfer". In: *arXiv preprint arXiv:2312.03673* (2023).

[49] T. Matiisen, A. Oliver, T. Cohen, and J. Schulman. "Teacher–student curriculum learning". In: *IEEE transactions on neural networks and learning systems* 31.9 (2019), pp. 3732–3740.

[50] R. Portelas, C. Colas, L. Weng, K. Hofmann, and P.-Y. Oudeyer. "Automatic curriculum learning for deep rl: A short survey". In: *arXiv preprint arXiv:2003.04664* (2020).

[51] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. "Learning quadrupedal locomotion over challenging terrain". In: *Science robotics* 5.47 (2020), eabc5986.

[52] W. Yu, C. Yang, C. McGreavy, E. Triantafyllidis, G. Bellegarda, M. Shafiee, A. J. Ijspeert, and Z. Li. "Identifying important sensory feedback for learning locomotion skills". In: *Nature Machine Intelligence* 5.8 (2023), pp. 919–932.

[53] J. Parker-Holder, R. Rajan, X. Song, A. Biedenkapp, Y. Miao, T. Eimer, B. Zhang, V. Nguyen, R. Calandra, A. Faust, et al. "Automated reinforcement learning (autorl): A survey and open problems". In: *Journal of Artificial Intelligence Research* 74 (2022), pp. 517–568.

[54] M. Kiran and M. Ozyildirim. "Hyperparameter tuning for deep reinforcement learning applications". In: *arXiv preprint arXiv:2201.11182* (2022).

[55] O. Khatib. "The operational space formulation in the analysis, design, and control of robot manipulators". In: *Preprints 3rd International Symposium of Robotics Re-search, Gouvieux (Chantilly), France, October*. 1985, pp. 7–11.

[56] N. Hogan. "Impedance control: An approach to manipulation". In: *1984 American control conference*. IEEE. 1984, pp. 304–313.

[57] S. Singh, R. L. Lewis, A. G. Barto, and J. Sorg. "Intrinsically motivated reinforcement learning: An evolutionary perspective". In: *IEEE Transactions on Autonomous Mental Development* 2.2 (2010), pp. 70–82.

[58] S. Niekum, A. G. Barto, and L. Spector. "Genetic programming for reward function search". In: *IEEE Transactions on Autonomous Mental Development* 2.2 (2010), pp. 83–90.

[59] A. Gupta, A. Pacchiano, Y. Zhai, S. M. Kakade, and S. Levine. *Unpacking Reward Shaping: Understanding the Benefits of Reward Engineering on Sample Complexity*. 2022. arXiv: 2210.09579 [cs.LG].

[60] J. Eschmann. "Reward function design in reinforcement learning". In: *Reinforcement Learning Algorithms: Analysis and Applications* (2021), pp. 25–33.

[61] D. Chen, B. Zhou, V. Koltun, and P. Krähenbühl. "Learning by cheating". In: *Conference on Robot Learning*. PMLR. 2020, pp. 66–75.

[62] G. B. Margolis and P. Agrawal. "Walk these ways: Tuning robot control for generalization with multiplicity of behavior". In: *Conference on Robot Learning*. PMLR. 2023, pp. 22–31.

[63] X. B. Peng, Z. Ma, P. Abbeel, S. Levine, and A. Kanazawa. "Amp: Adversarial motion priors for stylized physics-based character control". In: *ACM Transactions on Graphics (ToG)* 40.4 (2021), pp. 1–20.

[64] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. "Curiosity-driven exploration by self-supervised prediction". In: *International conference on machine learning*. PMLR. 2017, pp. 2778–2787.

[65] N. Rudin, D. Hoeller, P. Reist, and M. Hutter. "Learning to walk in minutes using massively parallel deep reinforcement learning". In: *Conference on Robot Learning*. PMLR. 2022, pp. 91–100.

[66] T. Chen, J. Xu, and P. Agrawal. "A system for general in-hand object re-orientation". In: *Conference on Robot Learning*. PMLR. 2022, pp. 297–307.

[67] G. B. Margolis, G. Yang, K. Paigwar, T. Chen, and P. Agrawal. "Rapid locomotion via reinforcement learning". In: *arXiv preprint arXiv:2205.02824* (2022).

[68] P. Goyal, S. Niekum, and R. J. Mooney. "Using natural language for reward shaping in reinforcement learning". In: *arXiv preprint arXiv:1903.02020* (2019).

[69] Y. J. Ma, W. Liang, G. Wang, D.-A. Huang, O. Bastani, D. Jayaraman, Y. Zhu, L. Fan, and A. Anandkumar. "Eureka: Human-level reward design via coding large language models". In: *arXiv preprint arXiv:2310.12931* (2023).

[70] T. Xie, S. Zhao, C. H. Wu, Y. Liu, Q. Luo, V. Zhong, Y. Yang, and T. Yu. "Text2Reward: Automated Dense Reward Function Generation for Reinforcement Learning". In: *arXiv preprint arXiv:2309.11489* (2023).

[71] Y. Hu, W. Wang, H. Jia, Y. Wang, Y. Chen, J. Hao, F. Wu, and C. Fan. "Learning to utilize shaping rewards: A new approach of reward shaping". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 15931–15941.

[72] Z. Xian, T. Gervet, Z. Xu, Y.-L. Qiao, and T.-H. Wang. "Towards A Foundation Model for Generalist Robots: Diverse Skill Learning at Scale via Automated Task and Scene Generation". In: *arXiv preprint arXiv:2305.10455* (2023).

[73] Y. Yang, F.-Y. Sun, L. Weihs, E. VanderBilt, A. Herrasti, W. Han, J. Wu, N. Haber, R. Krishna, L. Liu, et al. "Holodeck: Language Guided Generation of 3D Embodied AI Environments". In: *arXiv preprint arXiv:2312.09067* (2023).

[74] Y. Kim, H. Oh, J. Lee, J. Choi, G. Ji, M. Jung, D. Youm, and J. Hwangbo. "Not only rewards but also constraints: Applications on legged robot locomotion". In: *arXiv preprint arXiv:2308.12517* (2023).

[75] Z. Zheng, J. Oh, and S. Singh. "On learning intrinsic rewards for policy gradient methods". In: *Advances in Neural Information Processing Systems* 31 (2018).

[76] E. Chen, Z.-W. Hong, J. Pajarinen, and P. Agrawal. "Redeeming intrinsic rewards via constrained optimization". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 4996–5008.

[77] J. A. Häusser, S. Schulz-Hardt, T. Schultze, A. Tomaschek, and A. Mojzisch. "Experimental evidence for the effects of task repetitiveness on mental strain and objective work performance". In: *Journal of Organizational Behavior* 35.5 (2014), pp. 705–721.

[78] E. Todorov, T. Erez, and Y. Tassa. "MuJoCo: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[79] S. Caron, Y. De Mont-Marin, R. Budhiraja, S. H. Bang, I. Domrachev, and S. Nedelchev. *Pink: Python inverse kinematics based on Pinocchio*. Version 3.0.0. 2024. URL: https://github.com/stephane-caron/pink.

[80] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. "Learning fine-grained bimanual manipulation with low-cost hardware". In: *arXiv preprint arXiv:2304.13705* (2023).

[81] K. Zakka, Y. Tassa, and MuJoCo Menagerie Contributors. *MuJoCo Menagerie: A collection of high-quality simulation models for MuJoCo*. 2022. URL: http://github.com/google-deepmind/mujoco_menagerie.

[82] URL: https://support.apple.com/en-us/118507#:~:text=Apple%20Vision%20Pro%20features%20an,feel%20contact%20on%20your%20nose..

[83] M. Daneshmand, A. Helmi, E. Avots, F. Noroozi, F. Alisinanoglu, H. S. Arslan, J. Gorbova, R. E. Haamer, C. Ozcinar, and G. Anbarjafari. "3d scanning: A comprehensive survey". In: *arXiv preprint arXiv:1801.08863* (2018).

[84] S. Hampali, T. Hodan, L. Tran, L. Ma, C. Keskin, and V. Lepetit. "In-hand 3d object scanning from an rgb sequence". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 17079–17088.

[85] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. "Describing textures in the wild". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 3606–3613.

[86] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, A. Kolesnikov, et al. "The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale". In: *International journal of computer vision* 128.7 (2020), pp. 1956–1981.

[87] A. Quattoni and A. Torralba. "Recognizing indoor scenes". In: *2009 IEEE conference on computer vision and pattern recognition*. IEEE. 2009, pp. 413–420.

[88] A. Xie, L. Lee, T. Xiao, and C. Finn. "Decomposing the generalization gap in imitation learning for visual robotic manipulation". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 3153–3160.